AA.VV.

commodore 128

LA GRANDE GUIDA DEL PROGRAMMATORE

- BASIC 7.0
- GRAFICA
- LINGUAGGIO MACCHINA
- SPRITE
- PROGRAMMAZIONE 8563
- SUONO E MUSICA
- INPUT/OUTPUT
- SISTEMA OPERATIVO
- CP/M 3.0
- MAPPA DI MEMORIA
- SPECIFICHE HARDWARE
- SCHEMI ELETTRICI, TABELLE, PRONTUARI

GRUPPO EDITORIALE JACKSON

commodore 128

LA GRANDE GUIDA

DEL PROGRAMMATORE

AA.VV.

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

PREFAZIONE

L'enciclopedia, nata con la collaborazione di tutti i tecnici responsabili della Commodore, si presenta come la guida di riferimento ufficiale per quello che si può considerare come uno dei più versatili e interessanti computer attualmente disponibili. Con 128K di memoria, espandibili a 256K fino ad un massimo di 640K, schermo a 80 colonne, compatibilità hardware e software, funzionamento in modo CP/m 3.0, linguaggio BASIC espanso: queste tra le più significative. Altra importante caratteristica particolare di questa macchina sta nell'essere riuscita a racchiudere in un solo calcolatore tre diversi modi operativi primari:

*- **Modo C128** Il computer fornisce la capacità di memoria richiesta da applicazioni sofisticate quali trattamento testi, fogli elettronici e programmi gestione dati;*

*- **Modo C64** In questa configurazione il computer permette di essere utilizzato sfruttando l'ampia gamma di software reperibile sul mercato;*

*- **Modo CP/M** Un processore Z80 incorporato dà accesso alle possibilità del CP/M 3.0 della Digital Research, più altre capacità innovative aggiunte dalla Commodore.*

Questa guida, dunque, ha l'intento di essere sfruttata per riuscire ad usare nella maniera più consona tutte le potenzialità di questo particolare calcolatore, venendo a conoscenza di tutti i suoi più nascosti meccanismi.

INDICE

Appendici

RINGRAZIAMENTI

Scritto da Larry Greenley
e
Fred Bowen
Bil Herd
Dave Haynie
Terry Ryan
Von Ertwine
Kim Eckert
Mario Eisenbacher
Norman McVey

Gli autori sono molto in debito con tutte le persone che hanno contribuito alla stesura di questo libro. Ringraziamo particolarmente Jim Gracely delle Pubblicazioni Commodore, che ha rivisto l'intero manoscritto nella parte tecnica e fornito importanti correzioni, spiegazioni e suggerimenti per l'utente, ed anche Steve Beats e Dave Middleton della sezione software Commodore per il loro aiuto e la loro esperienza nella stesura dei programmi.

Vogliamo riconoscere ancora il contributo dato da Frank Palaia della sezione di progettazione hardware Commodore, che ha fornito la sua esperienza nell'hardware Z80, e quello di Dave DiOrio della sezione di progettazione circuiti Commodore, che ha disegnato l'Unità di Gestione della Memoria e i miglioramenti del chip VIC II del C128.

Grazie anche a Bob Albright, Pete Bowman, Steve Lam e Tony Porrazza della sezione progettazione per le numerose correzioni tecniche del manoscritto. Ringraziamenti a Dan Baker, Dave Street e Carolyn Scheppner per il supporto tecnico del software Commodore e per essere sempre stati disponibili nell'assistenza tecnica. Inoltre, desideriamo riconoscere il notevole contributo dato dai membri del controllo di qualità del software Commodore, in special modo Mike Colligon, Karen Mackenzie, Pat McAllister, Greg Rapp, Dave Resavy e Stacy English.

1

INTRODUZIONE

Il personal computer Commodore 128 è un versatile computer multimodo. Il Commodore 128 è il successore del vendutissimo Commodore 64. Le principali caratteristiche del Commodore 128 sono:

- 128 Kbyte di RAM, espandibili a 256K o 640K
- schermo a 80 colonne
- compatibilità hardware e software col Commodore 64
- funzionamento in modo CP/M 3.0
- linguaggio BASIC espanso

Come mostrerà questa guida, il Commodore 128 ha molte altre nuove ed estese capacità e caratteristiche. Quelle sopra riassunte, comunque, sono le più significative nel confrontare le possibilità del Commodore 128 con quelle del Commodore 64 e di altri microcomputer.

Il Commodore 128 in realtà racchiude tre computer in uno, con i seguenti tre modi operativi primari:

- Modo C128
- Modo C64
- Modo CP/M

Due di questi modi primari (C128 e CP/M) possono operare usando sia uno schermo a 40 che a 80 colonne. Ecco un compendio delle principali caratteristiche dei tre modi operativi primari.

MODO C128

Nel Modo C128, il personal computer Commodore 128 fornisce la capacità di memoria richiesta da applicazioni sofisticate, quali trattamento testi, fogli elettronici e programmi di gestione dati.

Le caratteristiche del Modo C128 comprendono:

- processore 8502 funzionante a 1.02 o 2.04 MHz
- un nuovo ed espanso Kernal C128
- monitor di linguaggio macchina incorporato
- linguaggio BASIC 7.0 Commodore, con oltre 140 comandi e funzioni
- nuovi ed appositi comandi BASIC 7.0 per creare meglio, più velocemente e facilmente grafica complessa, animazione, suono e programmi musicali
- uscita per 40 colonne di testo e grafica bit map col chip VIC II
- uscita per 80 colonne di testo col chip 8563

NOTA: gli schermi a 40 ed 80 colonne possono essere usati sia singolarmente che contemporaneamente usando due monitor.

- suono (tre voci) col chip SID
- una tastiera a 92 tasti incluso un tastierino numerico e i tasti ESCAPE, TAB, ALT, CAPS LOCK, HELP, LINE FEED, 40/80 DISPLAY e NO SCROLL.
- pieno accesso alle capacità delle nuove periferiche Commodore (drive veloce 1571, monitor duale 1902 a 40/80 colonne, ecc.)
- accesso a tutte le normali periferiche Commodore
- espansione della RAM a 256K o 640K con i moduli di espansione RAM

MODO C64

Nel Modo C64, il Commodore 128 possiede tutte le capacità del Commodore 64, permettendo di usare l'ampia gamma di software disponibile per questo computer.

Le caratteristiche del Modo C64 comprendono:

- processore 8502 funzionante a 1.02 MHz
- Kernal standard C64
- linguaggio BASIC 2.0
- 64K di RAM
- uscita video a 40 colonne col chip VIC II
- suono (tre voci) col chip SID
- tastiera standard a 64 tasti del Commodore 64, eccetto che per i tasti funzione
- tutte le normali funzioni di tastiera del Commodore 64
- accesso a tutte le capacità grafiche e sonore del Commodore 64
- compatibilità con le periferiche standard del Commodore 64, comprese le porte utente e seriale, il registratore Datassette (tm), joystick, monitor video compositi e uscita RF (TV)

NOTA: il drive 1571 funzionerà in Modo C64, ma solo alla normale velocità 1541. Ciò è richiesto per mantenere la compatibilità col vero C64.

MODO CP/M

Nel Modo CP/M, un processore Z80 incorporato dà accesso alle possibilità del CP/M 3.0 della Digital Research, più altre nuove capacità aggiunte dalla Commodore.

Le caratteristiche del Modo CP/M comprendono:

- processore Z80 incorporato funzionante a 2.04 MHZ
- sistema CP/M 3.0 su disco
- 128 Kbyte di RAM (in banchi di 64K)
- uscita video a 40 colonne col chip VIC II
- uscita video a 80 colonne col chip 8563
- disponibilità di tutta la tastiera, compresi il tastierino e i tasti speciali
- accesso al nuovo drive veloce 1571 e a tutte le periferiche standard
- possibilità di ridefinire quasi ogni tasto
- possibilità di emulare parecchi terminali (Lear-Siegler ADM31, ADM 3A)
- uso di dischi di vario formato (IBM, Kaypro, Epson, Osborne)
- espansione RAM a 256K o 640K con i moduli di espansioneRAM

L'implementazione del CP/M 3.0 (detto anche CP/M Plus) sul Commodore128 rende disponibili all'utente parecchie migliaia di famosi programmi commerciali e di pubblico dominio.

COMPONENTI HARDWARE

Il Commodore 128 incorpora i seguenti componenti hardware:

PROCESSORI

8502: Processore principale nei Modi C128 e C64; supporto I/O al CP/M; compatibile col software per 6502; gira a 1.02 o 2.04 MHz
Z80: solo Modo CP/M; gira a 2.04 MHz

MEMORIA

ROM: standard 64K (Kernal e BASIC del C64; Kernal e BASIC del 128, ROM caratteri e BIOS del CP/M); uno slot di 32K per software esterno
RAM: 128K in due banchi di 64K; 16K di RAM video per il chip 8563; RAM colore di 2K X 4

VIDEO

8564: video a 40 colonne (versioni separate per gli standard TV PAL e NTSC)
8563: video a 80

SUONO

6581: chip SID

INPUT/OUTPUT

6526: porte joystick/scansione della tastiera, registratore
6526: porte utente e seriale

GESTIONE DELLA MEMORIA

8921: PLA (mappatura dei Modi C128 e C64)
8922: MMU (circuito logico esclusivo per C128)

Per dettagli su questi ed altri componenti hardware vedere nella VII parte, Specifiche Hardware del Commodore 128.

COMPATIBILITA' COL COMMODORE 64

Il Commodore 128 rappresenta un aggiornamento del Commodore 64. Così, una delle principali caratteristiche del Commodore 128 in Modo C64 è la piena compatibilità hardware e software col Commodore 64. Ciò significa che in Modo C64 il Commodore 128 può eseguire tutte le applicazioni per il Commodore 64. Inoltre, il Commodore 128 in Modo 64 può usare tutte le periferiche eccetto la cartuccia del CP/M 2.2. (NOTA: la capacità del CP/M 3.0 incorporato nel C128 supera quella fornita dalla cartuccia esterna. Questa cartuccia non deve essere usata in nessun modo col C128).

Il Modo C128 è visto come un'aggiunta compatibile al C64. Specificamente, tutte le funzioni del Kernal fornite dal Commodore 64 sono fornite anche nel Kernal del C128. Queste funzioni sono inoltre fornite alle stesse locazioni nella tabella dei salti del Kernal del C128 per mantenere la compatibilità con i programmi esistenti.

Le variabili di sistema in pagina zero ed altrove sono state mantenute agli stessi indirizzi che occupano nel Modo C64. Ciò semplifica l'adattamento di molti programmi. Fornire la compatibilità col Commodore 64 significa che le nuove caratteristiche del Commodore 128 non possono essere usate in Modo C64. Per esempio, i vincoli di compatibilità e di memoria precludono la modifica del Kernal del C64 per supportare il drive veloce 1571. Come visto prima, il Modo C64 vede questo drive come un drive seriale standard. Per la stessa ragione, il Modo C64 non ha un editor di schermo ad 80 colonne, e il BASIC.2.0 del Modo C64 non può usare il secondo banco di memoria da 64K.

COME PASSARE DA UN MODO ALL' ALTRO

Come menzionato prima, nei Modi C128 e CP/M il Commodore 128 può fornire sia schermi a 40 che a 80 colonne. Questo significa che il Commodore 128 ha in realtà cinque modi operativi, come segue:

- Modo C128 con video a 80 colonne
- Modo C128 con video a 40 colonne
- Modo C64 (solo video a 40 colonne)
- Modo CP/M con video a 80 colonne
- Modo CP/M con video a 40 colonne

La figura 1.1 elenca i metodi per passare da un modo all'altro.

DA	A					
	SPENTO	**C128 40 COL**	**C128 80 COL**	**C64**	**CP/M 40 COL**	**CP/M 80 COL**
C128 40 COL	1. Su il tasto 40/80 2. Leva i dischi CP/M o la cartuccia 3. Accendi il computer		1. Premi e rilascia ESC 2. Premi X Oppure 1. Su il tasto 40/80 2. Premi il pulsante di reset	1. Su il tasto 40/80 2. Spegni accendi il computer 3. Rimuovi la cartuccia	1. Su il tasto 40/80 2. Spegni e accendi	1. Su il tasto 40/80 2. Spegni e accendi
C128 80 COL	1. Giù il tasto 40/80 2. Accendi il computer	1. Premi e rilascia ESC 2. Premi X Oppure 1. Giù il tasto 40/80 2. Premi il pulsante di reset		1. Giù il tasto 40/80 2. Spegni e accendi il computer 3. Leva la cartuccia	1. Giù il tasto 40/80 2. Leva i dischi CP/M 3. Spegni e accendi il computer	1. Giù il tasto 40/80 2. Leva i dischi CP/M 3. Spegni e accendi il computer

Figura 1-1. Tavola di commutazione dei modi del Commodore 128.

DA	A					
	SPENTO	**C128 40 COL**	**C128 80 COL**	**C64**	**CP/M 40 COL**	**CP/M 80 COL**
C64	1. Giù il tasto C= 2. Accendi il computer Oppure 1. Metti una cartuccia 2. Accendi il computer	1. Scrivi GO64 e premi «Return» 2. Scrivi Y (YES) e premi «Return»	1. Scrivi GO64 e premi «Return» 2. Scrivi Y (YES) e premi «Return»		1. Spegni il computer 2. Su il tasto 40/80 3. Giù il tasto C= e accendi Oppure 1. Spegni il computer 2. Metti una cartuccia 3. Accendi il computer	1. Spegni il computer 2. Su il tasto 40/80 3. Giù il tasto C= e accendi Oppure 1. Spegni il computer 2. Metti una cartuccia 3. Accendi il computer
CP/M 40 COL	1. Accendi il drive 2. Metti il disco CP/M 3. Su il tasto 40/80 4. Accendi il computer	1. Accendi il drive 2. Metti il disco CP/M 3. Su il tasto 40/80 4. Scrivi BOOT 5. Premi «Return»	1. Accendi il drive 2. Metti il disco CP/M 3. Su il tasto 40/80 4. Scrivi BOOT 5. Premi «Return»	1. Su il tasto 40/80 2. Accendi il drive 3. Metti il disco CP/M 4. Spegni e accendi il computer		1. Metti il disco CP/M 2. Scrivi DEVICE Conout:= 40 COL 3. Premi «Return»
CP/M 80 COL	1. Accendi il drive 2. Metti il disco CP/M 3. Giù il tasto 40/80 4. Accendi il computer	1. Accendi il drive 2. Metti il disco CP/M 3. Giù il tasto 40/80 4. Scrivi BOOT 5. Premi «Return»	1. Accendi il drive 2. Metti il disco P/M 3. Giù il tasto 40/80 4. Scrivi BOOT 5. Premi «Return»	1. Giù il tasto 40/80 2. Accendi il drive 3. Metti il disco CP/M 4. Spegni e accendi il computer	1. Metti il disco CP/M 2. Scrivi DEVICE Conout:= 80 COL 3. Premi «Return»	

Figura 1-1. Tavola di commutazione dei modi del Commodore 128.

NOTA: Se state usando il monitor duale Commodore 1902 ricordate di spostare l'interruttore sul monitor da COMPOSITO o SEPARATO a RGBI quando passate da 40 a 80 colonne, e viceversa da 80 a 40. Inoltre, quando alternate fra vari modi, rimuovete ogni cartuccia dalla porta di espansione. Potreste anche aver bisogno di levare un disco (per esempio, CP/M) dal drive.

VERSIONI DEL SISTEMA CP/M 3.0

Quando spedite la garanzia del vostro C128 il vostro nome verrà aggiunto ad una lista in modo da potervi inviare le successioni versioni del sistema CP/M.

2

FONDAMENTALI DEL BASIC

Il linguaggio BASIC si compone di comandi, operatori, costanti, variabili, vettori e stringhe. I comandi sono istruzioni seguite dal computer per effettuare un'operazione. Gli altri elementi del BASIC svolgono varie funzioni, come assegnare valori ad una quantità, passare valori al computer, od indirizzare il computer nello svolgimento di operazioni matematiche. Questa sezione descrive la struttura e le funzioni degli elementi del linguaggio BASIC.

COMANDI ED ISTRUZIONI

Per definizione, comandi ed istruzioni si distinguono come segue. Un comando è una parola BASIC usata in modo immediato. Non è preceduto da un numero di linea e viene eseguita immediatamente dopo aver premuto il tasto RETURN. Un'istruzione è una parola BASIC contenuta in un programma ed è preceduta da un numero di linea. Le istruzioni di un programma sono eseguite tramite il comando RUN seguito dal tasto RETURN. Molti comandi possono essere eseguiti da programma. In questo caso il comando è preceduto da un numero di linea e si dice che è usato in modo programma. Parecchi comandi possono essere usati anche al di fuori di un programma in quello che è chiamato il modo diretto. Per esempio, LOAD è un comando spesso usato in modo diretto, ma potete anche includerlo in un programma. GET e INPUT sono comandi che possono essere usati solo in un programma; altrimenti si incorre in un ILLEGAL DIRECT ERROR (errore di modo diretto non consentito). Mentre PRINT è solitamente incluso in un programma, potete usarlo anche in modo diretto per far stampare un messaggio od un valore numerico sullo schermo, come nell'esempio seguente:

PRINT "IL COMMODORE 128" **«RETURN»**

Notate che il messaggio è mostrato sullo schermo non appena premete il tasto RETURN. Le due linee seguenti mostrano lo stesso messaggio sullo schermo. La prima è un'istruzione in modo programma; la seconda è un comando in modo diretto.

10 PRINT "IL COMMODORE 128" **«RETURN»**

RUN **«RETURN»**

E' importante conoscere i concetti che stanno dietro alla memorizzazione dei dati prima di esaminare il linguaggio BASIC Commodore in dettaglio. Specificamente, dovete conoscere costanti, variabili e vettori.

MEMORIZZAZIONE DEI DATI NUMERICI: COSTANTI, VARIABILI E VETTORI

Ci sono tre modi di memorizzare dati col BASIC Commodore. Il primo è usare

una costante. Una costante è una forma di memorizzazione nella la quale i contenuti rimangono gli stessi lungo tutto il corso del programma. Il secondo modo utilizza una variabile. Come indica il nome, una variabile è una zona di memoria nella quale i contenuti variano o cambiano durante lo svolgimento del programma. L'ultimo modo di memorizzare informazioni è servirsi di un vettore, cioè di una serie di locazioni di memoria collegate fra loro tramite un indice, usate come delle variabili. A ognuna di queste tre differenti unità di memorizzazione possono essere assegnati tre diversi tipi di dati. Questi tre tipi sono INTERO, REALE e STRINGA. I dati interi sono numerici, senza parte decimale. I reali includono invece una parte decimale separata da quella intera con un punto. Il tipo stringa rappresenta invece una serie di simboli alfanumerici a cui ci si riferisce come un'unica parola. I paragrafi seguenti descrivono questi tre tipi di dati e il modo in cui essi vengono memorizzati.

COSTANTI: INTERE, REALI E STRINGA

COSTANTI INTERE

Il valore assegnato ad una costante rimane immodificato (costante) durante un programma. Le costanti intere possono contenere un valore positivo o negativo compreso da −32768 a +32767. Se il segno più è omesso il Commodore 128 assume l'intero come positivo. Le costanti intere non contengono virgole o punti fra le cifre. Gli zeri in testa vengono ignorati. Gli interi vengono memorizzati come numeri binari a due byte, il che significa che una costante richiede 16 bit o due byte di memoria per immagazzinare l'intero come numero in base due. Ecco alcuni esempi di costanti intere:

1
1000
−32
0
−32767

COSTANTI REALI

Le costanti reali (dette anche in virgola mobile) contengono una parte frazionaria indicata dal punto decimale. Non vi appaiono virgole per separare le cifre. Queste costanti possono essere positive o negative. Se si omette il segno più vengono considerte positive. Di nuovo, gli zeri in testa vengono ignorati. Le costanti reali sono rappresentate in due modi a seconda del loro valore:

1. Notazione Numerica Semplice
2. Notazione Scientifica

Nella notazione numerica semplice, il numero reale è calcolato fino a dieci cifre e memorizzato usando cinque byte, ma vengono stampate solo nove cifre. Se il numero reale è più grande di nove cifre, la nona viene arrotondata rispetto alla decima. L'arrotondamento avviene per difetto se la decima cifra è minore di cinque, altrimenti per eccesso. L'arrotondamento può divenire un fattore da considerare nell'eseguire calcoli maggiori di nove cifre, e i vostri programmi

dovrebbero quantificare l'errore causato da tale situazione. Come detto,i numeri reali sono mostrati a nove cifre.

Se il valore della costante reale è minore di .01 o maggiore di 999999999, il numero è mostrato in notazione scientifica. Per esempio, il numero 12345678901 è mostrato come 1.23456789E+10. Altrimenti viene usata la notazione semplice. Una costante reale in notazione scientifica appare distinta in tre parti:

1. La mantissa è il numero a sinistra separato dal punto decimale.
2. La lettera E indica che il numero è mostrato in notazione esponenziale (scientifica).
3. L'esponente specifica la potenza di dieci alla quale il numero viene elevato e quindi il numero di posti di cui viene spostato il punto per rappresentare il numero in notazione semplice.

La mantissa e l'esponente possono essere positivi o negativi. L'esponente può variare nel raggio da -39 a +38. Se l'esponente è negativo, il punto decimale si sposta a sinistra, altrimenti a destra. Il Commodore 128 limita la grandezza dei numeri reali. Il più alto numero rappresentabile in notazione esponenziale è 1.70141183E+38. Se provate a rappresentare un numero più grande incorrerete in un OVERFLOW ERROR (errore di trabocco). Il più piccolo numero rappresentabile in notazione scientifica è 2.93873588E-39. Se provate a rappresentare un numero più piccolo non verrà segnalato nessun errore e verrà ritornato uno zero. Dovreste quindi sempre controllare i valori reali nei vostri programmi se i vostri calcoli si basano su numeri molto piccoli, usati più volte.

Ecco degli esempi di costanti reali in notazione semplice e scientifica:

semplice	scientifica
9.99	22.33E+20
.0234	99999.234E-23
+10.01	-45.89E-11
−90.23	-3.14E+17

NOTA: i valori nelle due colonne non sono equivalenti.

COSTANTI STRINGA

Una costante stringa, come detto, è una serie di caratteri alfanumerici (numeri, lettere e simboli). Una costante stringa può essere lunga quanto una linea di input di 160 caratteri, meno il numero di linea ed altre informazioni che appaiono sulla stessa linea. Concatenando insieme le stringhe potete formare una stringa lunga fino a 255 caratteri. La stringa può contenere numeri, lettere, ed anche punti decimali e virgole. Comunque, la stringa non può contenere le virgolette (''), poichè esse delimitano l'inizio e la fine della stringa. Potete rappresentare questo carattere usando CHR$(34). Potete omettere le virgolette finali in una stringa se questa è l'ultima istruzione in una linea di programma.

A una stringa può anche essere assegnato un valore nullo, che significa che essa non contiene alcun carattere. Per assegnare un valore nullo, aprite e chiudete subito le virgolette. Ecco alcuni esempi di costanti stringa:

```
"COMMODORE 128"
"QWER1234!#$%()*.:,"
"" (stringa nulla)
"JOHN e JOAN"
```

VARIABILI: INTERE, REALI E STRINGA

Le variabili sono unità di memorizzazione che rappresentano dati variabili durante un programma. Al contrario delle costanti, le variabili possono cambiare di valore nel corso del programma. Il valore assegnato ad una variabile può essere un intero, un reale o una stringa. Potete assegnare ad una variabile il risultato di un calcolo matematico. Assegnate valori ad una variabile tramite il segno di uguale (=). Il nome della variabile appare a sinistra del segno, mentre il valore da assegnare appare sulla destra. Quando, in un programma, vi riferite ad una variabile prima di averle assegnato un valore, questa variabile varrà zero se di tipo intero o reale. Se di tipo stringa sarà invece nulla.

I nomi delle variabili possono avere qualsiasi lunghezza, ma per usarle con efficacia dovreste limitare il nome a pochi caratteri. Solo i primi due caratteri del nome di una variabile sono significativi. Quindi, non iniziate i nomi di due variabili diverse con gli stessi due caratteri. Se lo fate, il C128 le tratterà come un'unica variabile.

Il primo carattere del nome di una variabile deve essere una lettera. Il resto della variabile possono essere lettere o cifre da 0 a 9. Un nome di variabile non può contenere una parola chiave del BASIC. Se la contiene otterrete un SYNTAX ERROR (errore di sintassi). Le parole chiave del BASIC includono istruzioni, comandi, nomi di funzioni e di operatori logici, variabili riservate.

Potete specificare il tipo di dato di una variabile facendone seguire il nome da un simbolo di percento (%) se la variabile è intera, da un simbolo di dollaro ($) se è una stringa. Se non specificate nessun carattere, il C128 assume che la variabile è reale. Ecco alcuni esempi di variabili con le relative assegnazioni:

```
      A = 3.679 (reale)
     Z% = 714 (intera)
     F$ = "EVVIVA IL COMMODORE 128" (stringa)
      T = A + Z% (reale)
 COUNT% = COUNT% + 1 (intera)
     G$ = "RICERCA DI UN MAGGIOR LIVELLO DI COSCIENZA"
          (stringa)
     H$ = F$ + G$ (stringa)
```

VETTORI: INTERI, REALI E STRINGA

Sebbene i vettori siano stati definiti, all'inizio di questo capitolo, come una serie di costanti o variabili collegate, voi dovete riferirvi a loro tramite un singolo nome di variabile, sia intera, reale o stringa. Tutti gli elementi sono dello stesso tipo del nome di variabile. Per accedere ad ogni elemento entro il vettore, il BASIC usa degli indici (variabili indicizzate) per riferirsi ad ogni singolo elemento del vettore. Per esempio, l'alfabeto ha ventisei lettere. Ipotizziamo che sia stato definito un vettore chiamato "ALFA" comprendente tutte le lettere dell'alfabeto. Per accedere al primo elemento del vettore, che è anche la prima lettera dell'alfabeto (A), date ad ALFA un indice di zero:

ALFA$ (0) A
Per accedere alla lettera B, date ad ALFA un indice di uno:

ALFA$ (1) B
Continuate nello stesso modo per accedere a tutti gli elementi del vettore ALFA, come adesso:

ALFA$ (2) C
ALFA$ (3) D
ALFA$ (4) E
ALFA$ (5) Z

Gli indici sono convenienti per accedere ad ogni singolo elemento nel vettore. Se gli indici non esistessero, dovreste assegnare ogni valore ad una variabile diversa. Il primo indice di un vettore è zero.

Sebbene i vettori siano memorizzati sequenzialmente in memoria, possono essere multidimensionali. Tavole e matrici sono manipolate molto facilmente coi vettori bidimensionali. Per esempio, fate conto di avere una matrice con dieci righe e dieci colonne. Avete bisogno di 100 locazioni o vettori così da memorizzare l'intera matrice. Anche se la vostra matrice è dieci per dieci, gli elementi nel vettore sono memorizzati uno dopo l'altro per 100 locazioni.

Usate DIM per specificare il numero di dimensioni nel vettore. Per esempio:

10 DIM A(99)
dimensiona un vettore reale monodimensionale di 100 elementi. Ecco degli esempi di vettori bi, tri e quadridimensionali:

```
20 DIM B(9,9)              (100 elementi)
30 DIM C(20,20,20)         (9261 elementi)
40 DIM D(10,15,15,10)      (30976 elementi)
```

In teoria il massimo numero di dimensioni di un vettore è 255, ma non potete scrivere un'istruzione DIM più lunga di 160 caratteri: quindi il massimo numero di dimensioni assegnabili è circa 50. Ogni dimensione può avere poi un numero di elementi fino a 32767. In pratica la grandezza di un vettore è limitata dalla memoria disponibile. Molti vettori sono mono, bi o tridimensionali. Se un vettore contiene complessivamente meno di dieci elementi non è necessario usare l'istruzione DIM, poichè il C128 creerà automaticamente lo spazio per dieci elementi. La prima volta che vi riferite ad un vettore non dimensionato il C128 gli assegnerà zero o una stringa nulla a seconda del tipo.

Nell'istruzione DIM dovete separare l'indice di ogni dimensione con una virgola. Gli indici possono essere costanti intere, variabili o il risultato intero di un'operazione. I valori leciti per gli indici vanno da zero al limite massimo dimensionato. Se l'indice eccede questo valore, si avrà un BAD SUBSCRIPT ERROR (letteralmente: errore per cattivo indice).

Il tipo di vettore determina quanta memoria è usata per memorizzare i dati interi, reali o stringa. Vettori reali e stringa occupano la maggior quantità di memoria, al contrario dei vettori interi. Ecco ora l'occupazione di memoria per ogni tipo di vettore:

+ 5 byte per il nome
+ 2 byte per ogni dimensione
+ 2 byte per ogni elemento intero
OPPURE + 5 byte per ogni elemento reale
OPPURE + 3 byte per ogni elemento stringa
E + 1 byte per ogni carattere nella stringa

Tenete bene in mente l'occupazione di ogni tipo di vettore. Se avete solo bisogno di un vettore intero specificatelo nell'istruzione DIM, poichè un vettore reale occupa molto più spazio.

Ecco alcuni esempi di vettori:

A$(0)="VENDITE ALL'INGROSSO"	(vettore stringa)
MTH$(K%)="GEN"	(vettore stringa)
G2%(X)=5	(vettore intero)
CNT%(G2%(X))=CNT%(1)-2	(vettore intero)
FP(12*K%)=24.8	(vettore reale)
SOM(CNT%(1))=FP*K%	(vettore reale)
A(5)=0	Pone il quinto elemento del vettore A uguale a 0
B(5,6)=26	Pone l'elemento nella riga 5, colonna 6 del vettore B uguale a 26
C(1,2,3)=100	Pone l'elemento nella riga 1, colonna 2 e profondità 3 del vettore C uguale a 100

ESPRESSIONI ED OPERATORI

Le espressioni sono composte usando costanti, variabili e/o vettori. Un'espressione può essere una singola costante, una semplice variabile o un vettore di qualsiasi tipo. Può anche essere una combinazione di costanti e variabili con operatori aritmetici, logici o relazionali in modo da produrre un unico valore. Il modo di lavorare degli operatori è spiegato più avanti. Le espressioni possono essere divise indue classi:

1. ARITMETICHE
2. STRINGA

Le espressioni hanno due o più elementi di dati chiamati operandi. Ogni operando è separato da un singolo operatore per produrre il risultato desiderato. Questo è quello che si fa di solito per assegnare un valore ad una variabile.

Un operatore è uno speciale simbolo che l'interprete BASIC del vostro Commodore 128 riconosce come rappresentante un'operazione da eseguire sui dati costanti o variabili. Uno o più operatori, combinati con una o più variabili o costanti forma un'espressione. Gli operatori aritmetici, logici e relazionali sono riconosciuti dal BASIC del Commodore 128.

ESPRESSIONI ARITMETICHE

Le espressioni aritmetiche producono un valore intero o reale. Gli operatori aritmetici (+,-,*,/,) sono usati per eseguire rispettivamente l'addizione, la sottrazione, la moltiplicazione, la divisione e l'elevamento a potenza.

OPERAZIONI ARITMETICHE

Un operatore aritmetico definisce un'operazione aritmetica compiuta sui due operandi ai lati dell'operatore. Le operazioni aritmetiche sono effettuate tramite numeri reali. Gli interi sono convertiti nel formato reale prima di eseguire l'operazione, e viceversa il risultato.

ADDIZIONE (+)

Il segno più (+) specifica che l'operando a destra viene aggiunto a quello a sinistra.

ESEMPI:

2+2
A+B+C
X%+1
BR+10E-2

SOTTRAZIONE (-)

Il segno meno (-) specifica che l'operando a destra è sottratto dall'operando a sinistra.

ESEMPI:

4-1
100-64
A-B
55-142

Il meno può anche essere usato come meno unario, cioè il segno meno davanti ad un numero negativo. Ciò equivale a sottrarre il numero da 0.

ESEMPI:

-5
-9E4
-B
4-(-2) (lo stesso che 4+2)

MOLTIPLICAZIONE (*)

Un asterisco(*) specifica che l'operando a sinistra è moltiplicato per quello a destra.

ESEMPI:

100*2
50*0
A*X1
R%*14

DIVISIONE (/)

La barra (/) specifica che l'operando a sinistra è diviso per l'operando a destra.

ESEMPI:

10/2
6400/4
A/B
4E2/XR

ELEVAMENTO A POTENZA (↑)

La freccia in su specifica che l'operando a sinistra è elevato alla potenza specificata dall'operando a destra (esponente). Se l'operando a destra è 2, il numero a sinistra è elevato al quadrato; se l'esponente è 3, il numero a sinistra è elevato al cubo, ecc. L'esponente può essere un qualsiasi numero che dia poi un risultato reale valido.

ESEMPI:

2 ↑ 2	Equivalente a 2*2
3 ↑ 3	Equivalente a 3*3*3
4 ↑ 4	Equivalente a 4*4*4*4
AB ↑ CD	
3 ↑ -2	Equivalente a $\frac{1}{3} * \frac{1}{3}$

OPERATORI RELAZIONALI

Gli operatori relazionali (<,=,>,<=,>=,<>) sono principalmente usati per confrontare due operandi, ma producono anche un risultato aritmetico. Gli operatori relazionali e gli operatori logici (AND, OR, NOT), quando usati in un confronto, producono una valutazione aritmetica dell'espressione di tipo vero/falso. Se la relazione è vera si avrà come risultato -1, altrimenti 0. Ecco gli operatori relazionali:

<	MINORE DI
=	UGUALE A
>	MAGGIORE DI
<=	MINORE O UGUALE
>=	MAGGIORE O UGUALE
<>	DIVERSO DA

ESEMPI:

5-4=1	risultato vero (-1)
14>66	risultato falso (0)
15>=15	risultato vero (-1)

Gli operatori relazionali possono essere usati per confrontare le stringhe. A tale scopo, le lettere dell'alfabeto hanno l'ordine A<B<C<D, ecc. Le stringhe

sono comparate valutando le relazioni fra i caratteri corrispondenti da sinistra a destra (guardate le operazioni sulle stringhe).

ESEMPI:

"A"<"B" risultato vero (-1)
"X"="YY" risultato falso (0)
BB$<>CC$ risultato falso se sono identiche

Dati numerici possono essere assegnati o comparati solo con altri dati numerici; lo stesso vale per le stringhe. Altrimenti si avrà il messaggio TYPE MISMATCH (tipo scambiato). Gli operandi numerici sono valutati prima convertendoli nel formato reale, se necessario, poi eseguendo il confronto per ottenere il valore vero/falso.

Alla fine del confronto si ottiene un intero indipendentemente dal tipo di operando (numerico o stringa). Perciò un confronto fra due operandi può essere usato come un operando in una successiva operazione. Il risultato sarà 0 o -1: per questo bisognerà fare attenzione nell'usarlo con la divisione, poichè una divisione per 0 non è ammessa.

OPERATORI LOGICI

Gli operatori logici (AND, OR, NOT) possono essere usati per modificare il significato degli operatori relazionali od anche per produrre un risultato aritmetico. Gli operatori logici possono produrre risultati diversi da 0 e -1, sebbene ogni risultato diverso da zero sia considerato vero in un confronto vero/falso.

Gli operatori logici (chiamati qualche volta operatori Booleani) possono essere usati anche per compiere operazioni su singoli bit di due operandi. Ma quando usate l'operatore NOT questo agirà solo sull'operando a destra. Gli operandi devono essere compresi fra-32768 e +32767 (i reali sono convertiti in interi) e quindi anche il risultato sarà intero.

Le operazioni logiche sono eseguite con una corrispondenza bit a bit fra i due operandi. L'AND logico produce un bit 1 solo se entrambi i bit degli operandi sono 1. L'OR logico produce un bit 1 se almeno uno dei bit è 1.Il NOT logico è l'esatto complemento dei bit. Dunque se un bit=1 allora bit=0, e viceversa.

L'OR esclusivo (XOR) non possiede un operatore logico, ma viene eseguito come parte dell'istruzione WAIT o come funzione XOR. Se entrambi i bit degli operandi sono 1 allora il risultato sarà 0, altrimenti 1.

Le operazioni logiche sono definite da gruppi di istruzioni che, prese assieme, formano la "tabella della verità" Booleana, come mostrato nella Tavola 2-1.

L'operazione AND dà risultato 1 solo se entrambi i bit sono a 1:
1 AND 1 = 1
0 AND 1 = 0
1 AND 0 = 0
0 AND 0 = 0

L'operazione OR dà risultato 1 se almeno uno dei bit è a 1:
1 OR 1 = 1 0 OR 1 = 1
1 OR 0 = 1 0 OR 0 = 0

L'operazione NOT logicamente complementa ogni bit:
NOT 1 = 0
NOT 0 = 1

L'OR esclusivo (XOR) è una funzione (non un operatore logico):
1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0

Tavola 2-1 Tabella della verità Booleana

Gli operatori logici AND, OR e NOT specificano un'operazione aritmetica Booleana da eseguire su espressioni a due operatori su entrambi i lati dell'espressione. Nel caso di NOT viene considerato solo l'operando a destra. Le operazioni logiche (o aritmetica Booleana) sono eseguite solo dopo le operazioni aritmetiche e relazionali.

ESEMPI:

IF A=100 AND B=100 THEN10	(se sia A che B hanno il valore 100 allora il risultato è vero)
A=96 AND 32:PRINT A	(A=32)
IF A=100 OR B=100 THEN 20	(se A o B hanno il valore 100 il risultato è vero)
A=64 OR 32:PRINT A	(A=96)
X=NOT 96	(il risultato è -97 (complemento a 2))

GERARCHIA DELLE OPERAZIONI

Tutte le espressioni eseguono i differenti tipi di operazione seguendo una gerarchia fissa. Certe operazioni hanno una priorità più alta e vengono eseguite prima di altre. Il normale ordine delle operazioni può essere modificato includendo due o più operandi fra parentesi ( ), creando una "sottoespressione". Le parti dell'espressione racchiusa fra parentesi saranno ridotte ad un solo valore prima di eseguire i calcoli fuori delle parentesi. Quando usate le parentesi in un'espressione assicuratevi di averne un numero pari di aperte e chiuse, altrimenti otterrete un SYNTAX ERROR. Delle espressioni con operandi fra parentesi possono essere racchiuse esse stesse fra parentesi, creando così espressioni a più livelli. In questo caso si parla di nidificazione. Le parentesi possono essere nidificate nelle espressioni fino a dieci livelli - dieci parentesi aperte e dieci chiuse. L'espressione più interna sarà calcolata per prima. Ecco alcuni esempi di espressione:

```
A+B

C↑ (D+E)/2

((X-C↑ (D+E)/2)*10)+1

GG$>HH$

JJ$+"ANCORA"

K%=1 AND M<>X

K%=2 OR (A=B AND M <X)

NOT(D=E)
```

L'interprete BASIC esegue le operazioni sulle espressioni eseguendo prima le operazioni aritmetiche, poi quelle relazionali, infine quelle logiche. Sia gli operatori logici che quelli aritmetici hanno un ordine di precedenza (o gerarchia delle operazioni) fra di loro. Gli operatori relazionali invece vengono valutati da sinistra a destra e non hanno quindi gerarchia. Nel caso che in un'espressione compaiano più operatori con la stessa gerarchia, le operazioni vengono eseguite da sinistra a destra. La gerarchia viene rispettata anche fra parentesi. La gerarchia delle operazioni logiche e aritmetiche è mostrata nella tabella 2-2, dal primo all'ultimo. Nel caso di notazione scientifica questa viene calcolata con priorità assoluta.

OPERATORE	DESCRIZIONE	ESEMPIO
↑	Potenza	BASE ↑ ESP
-	Negazione	-A
*/	Moltiplicazione	AB * CD
	Divisione	EF / GH
+	Addizione	CNT + 2
-	Sottrazione	JK-PQ
> =<	Operatori	A < = B
NOT	Complemento NOT	NOT K%
AND	AND logico	JK AND 128
OR	OR logico	PQ OR 15

Tavola 2-2 Gerarchia delle operazioni eseguite nelle espressioni

OPERAZIONI SU STRINGHE

Le stringhe sono confrontate usando gli stessi operatori relazionali (=,<>,<=,>=,<,>) dei confronti numerici. La comparazione della stringa viene fatta prendendo un carattere alla volta (da sinistra a destra) da ciascuna stringa e calcolandone il codice: se i due codici sono uguali, anche i caratteri lo sono. Se invece differiscono, allora la stringa che contiene il carattere col codice minore è anch'essa minore rispetto all'altra stringa. La comparazione può andare avanti fino alla fine di una delle due stringhe: allora la stringa più corta sarà la minore. Nel caso invece le due stringhe siano del tutto identiche, come caratteri e lunghezza, allora il risultato sarà vero (-1). Nella comparazione vengono considerati anche gli spazi bianchi all'inizio ed alla fine della stringa. Tutti i confronti, di qualunque tipo siano i dati, danno sempre un risultato intero, anche fra stringhe. Per questo si può usare un confronto fra due stringhe in un calcolo: bisognerà solo fare attenzione alla divisione; il confronto può dare infatti 0 o -1, e la divisione per 0 è impossibile e dà errore.

ESPRESSIONI STRINGA

Le espressioni sono trattate come se le seguisse un implicito "<>0". Questo significa che se l'espressione è vera verrà eseguita la restante linea di programma; altrimenti l'esecuzione passerà alla prossima. Proprio come con i numeri, potete fare operazioni anche con le variabili stringa. L'unico operatore aritmetico su stringhe riconosciuto dal BASIC 7.0 è il segno più (+) che è usato per concatenare fra loro le stringhe. Quando concatenate le stringhe, la stringa a destra viene aggiunta a quella posta a sinistra, formando una terza stringa. Il risultato può essere stampato immediatamente, usato in un confronto o assegnato ad una variabile. Se provate a confrontare o eguagliare un termine stringa ed uno numerico allora il BASIC darà un errore di TYPE MISMATCH.Ecco alcuni esempi di espressione stringa e di concatenazione:

```
10 A$="NOME":B$="FILE"
20 NOM$=A$+B$                (produce la stringa "NOMEFILE")
30 RES$="NUOVO"+A$+B$                  (produce la stringa
                                           "NUOVONOMEFILE")
```

3

ENCICLOPEDIA DEL BASIC 7.0

ORGANIZZAZIONE DELL'ENCICLOPEDIA DEL BASIC 7.0

Questa sezione del secondo capitolo elenca gli elementi del linguaggio BASIC 7.0 in un formato enciclopedico. Fornisce un elenco abbreviato di regole (sintassi) del BASIC 7.0 del Commodore 128, insieme a una breve descrizione di ognuna. Il BASIC 7.0 comprende tutti gli elementi del BASIC 2.0.

I differenti tipi di operazioni BASIC sono elencati in settori individuali, come segue:

1. **Comandi** ed **istruzioni:** i comandi usati per redigere, memorizzare e cancellare programmi, e le istruzioni BASIC usate nelle linee numerate di un programma.
2. **Funzioni:** le funzioni stringa, numeriche e di stampa.
3. **Parole e simboli riservati:** le parole ed i simboli riservati per l'uso dal linguaggio BASIC 7.0, e che non possono essere usati per nessun altro scopo.

FORMATO DEI COMANDI E DELLE ISTRUZIONI

Le definizioni dei comandi e delle istruzioni in questa enciclopedia sono riportate nel seguente formato:

Nome del comando **AUTO**

Breve definizione Abilita/disabilita la numerazione automatica delle linee

Formato del comando **AUTO [# linea]**

Sul formato e l'uso Questo comando abilita la caratteristica di numerazione automatica delle linee. Questo facilita il lavoro d'inserimento dei programmi, battendo automaticamente il numero della linea al posto dell'utente. Non appena inserite una linea di programma premendo RETURN, il prossimo numero di linea viene stampato sullo schermo e il cursore viene posizionato due spazi a destra del numero. L'argomento [# linea] si riferisce all'incremento fra ogni linea. AUTO senza argomento disabilita la numerazione automatica, così come RUN. Questa istruzione deve essere usata solo in modo diretto (all'esterno di un programma).

ESEMPI:

Esempi

AUTO 10 Numera automaticamente le linee di programma a passi di 10.
AUTO 50 Numera le linee a passi di 50.
AUTO Disabilita la numerazione automatica delle linee.

La linea in grassetto che definisce il formato consiste di:

Le parti del comando o dell'istruzione scritte in maiuscolo devono essere digitate esattamente. Le parole che deve fornire l'utente, come ad esempio il nome del programma, sono invece in minuscolo.

Quando appaiono le virgolette ("": di solito racchiudono un nome di programma) l'utente deve includerle nell'identico formato dell'esempio.

Le parole chiave sono parole facenti parte del linguaggio BASIC. Sono la parte centrale di un comando o un'istruzione, e dicono al computer che specie di azione deve fare. Queste parole non possono essere usate come nomi di variabile. Una lista completa di parole e simboli riservati è pubblicata alla fine di questo capitolo.

Le parole chiave, dette anche parole riservate, appaiono in lettere maiuscole. Le parole chiave possono essere digitate usando la parola intera o l'abbreviazione convenzionale. (Una lista completa delle abbreviazioni è presente nell'Appendice I). La parola chiave od abbreviazione deve essere immessa correttamente per evitare un errore. I messaggi di errore del BASIC e del DOS sono elencati rispettivamente nelle appendici A e B.

Gli argomenti, chiamati anche parametri, appaiono in lettere minuscole. Gli argomenti sono complementari alle parole chiave fornendo loro informazioni specifiche. Per esempio, la parola chiave LOAD dice al computer di caricare un programma mentre l'argomento "nome del programma" dice al computer quale specifico programma caricare. Un secondo argomento specifica da quale drive caricare il programma. Gli argomenti includono nomi di file, variabili, numeri di linea, ecc.

Le parentesi quadre [] mostrano argomenti opzionali. L'utente può selezionarne qualcuno od anche nessuno, a seconda del bisogno.

Le parentesi angolari <> indicano che l'utente DEVE scegliere fra uno degli argomenti elencati.

Una barra verticale separa le varie voci in una lista di argomenti quando le scelte sono limitate a quelle voci. Quando la barra verticale appare fra parentesi quadre, le scelte sono limitate alle voci nella lista, ma l'utente può anche non scegliere nessuna voce. Se invece la barra verticale appare dentro le parentesi angolari, l'utente DEVE scegliere una delle voci elencate.

I puntini di sospensione ... significano che un opzione di un argomento può essere ripetuta più volte.

Le virgolette "" racchiudono le stringhe di caratteri, i nomi di file ed altre espressioni. Quando gli argomenti sono racchiusi fra virgolette, pure queste devono essere incluse nel comando. In questa enciclopedia le virgolette non sono usate per descrivere i formati; esse sono richieste come parte di un comando o di un'istruzione.

Le parentesi () Quando gli argomenti sono racchiusi fra parentesi tonde, esse devono essere incluse nel comando o istruzione. Pure le parentesi tonde non sono usate per descrivere i formati, ma sono richieste come parte del comando o istruzione.

Variabile si riferisce ad ogni nome valido di variabile BASIC, come X, A$, T%, ecc.

Espressione si riferisce ad ogni espressione BASIC valida, come A+B+2, .5*(X+3), ecc.

> **NOTA:** Per tutti i comandi DOS, le variabili ed espressioni devono essere racchiuse fra parentesi.

COMANDI ED ISTRUZIONI DEL BASIC

APPEND

Aggiunge dati in coda ad un file sequenziale.

APPEND#del file,"nome del file"[,Ddrive][<ON | ,>Udispositivo]

ESEMPI:

Append#8,"MIOFILE"	Apre il file logico 8 chiamato "MIOFILE" e si prepara ad accodarvi dei dati tramite l'istruzione PRINT#.
Append#7,(A$),D0,U9	Apre il file logico il cui nome si trova in A$, sul disco 0 dell'unità 9.

AUTO

Abilita/disabilita la numerazione automatica delle linee

AUTO[#linea]

ESEMPI:

AUTO 10	Numera automaticamente le linee di programma a passi di 10.
AUTO 50	Numera le linee a passi di 50.
AUTO	Disabilita la numerazione automatica.

BACKUP

Copia l'intero contenuto da un disco ad una altro con un drive doppio.

BACKUP Ddrive sorgente TO Ddrive destinazione[<ON | ,> Udispositivo]

> **NOTA:** Questo comando può essere usato solo con un drive a doppia testina.

ESEMPI:

BACKUP D0 TO D1	Copia tutti i file dal drive 0 al drive 1 sull'unità 8.
BACKUP D0 TO D1 ON U9	Copia tutti i file dal drive 0 al drive 1 sull'unità 9.

BANK

Seleziona uno fra 16 banchi BASIC (configurazioni di memoria), da 0 a 15, per l'uso coi comandi POKE, PEEK, SYS e WAIT.

BANK# di banco

Ecco una tavola delle configurazioni di memoria disponibili nella memoria del Commodore 128.

BANCO	CONFIGURAZIONE
0	solo RAM(0)
1	solo RAM(1)
2	solo RAM(2) (lo stesso che 0)
3	solo RAM(3) (lo stesso che 1)
4	ROM interna, RAM(0), I/O
5	ROM interna, RAM(1), I/O
6	ROM interna, RAM(2), I/O (lo stesso che 4)
7	ROM interna, RAM(3), I/O (lo stesso che 5)
8	ROM esterna, RAM(0), I/O
9	ROM esterna, RAM(1), I/O
10	ROM esterna, RAM(2), I/O (lo stesso che 8)
11	ROM esterna, RAM(3), I/O (lo stesso che 9)
12	Kernal e ROM interna bassa, RAM(0),I/O
13	Kernal e ROM esterna bassa, RAM (0), I/O
14	Kernal e ROM BASIC, RAM(0), ROM carattere
15	Kernal e ROM BASIC, RAM(0), I/O

BEGIN/BEND

Questa è un'istruzione condizionale tipo IF...THEN:ELSE, strutturata in modo da poter includere parecchie linee di programma fra l'inizio (BEGIN) e la fine (BEND) della struttura. Questo è il formato:

IF condizione THEN BEGIN: istruzione
istruzione
istruzione BEND: ELSE BEGIN
istruzione
istruzione BEND

ESEMPIO:

```
10 IF X=1 THEN BEGIN: PRINT"X=1 è vero"
20 PRINT" Così questa parte dell'istruzione viene eseguita"
30 PRINT"Quando x uguale a 1"
40 BEND: PRINT"Fine della struttura BEGIN/BEND":GO to 60
50 PRINT"X non è uguale a 1":PRINT"Ho saltato le istruzioni fra BEGIN/
   BEND"
60 PRINT"Resto del programma"
```

BLOAD

Carica un file binario iniziante ad una specifica locazione di memoria.

BLOAD"nome del file" [,Ddrive] [<ON | ,U>Udispositivo]
[,B# di banco] [,Pindirizzo iniziale]

dove:

- nome del file è il nome del vostro programma
- # di banco è la configurazione da 0 a 15
- indirizzo iniziale è la locazione d'inizio caricamento

ESEMPI:

BLOAD"SPRITE",B0,P3584	Carica il file binario di nome "SPRITE" a partire dalla locazione 3584 del banco 0.
BLOAD"DATA1",D0,U8,B1,P4096	Carica il file "DATA1" dalla locazione 4096 nel banco 1 dal disco 0, unità 8.

BOOT

Carica ed esegue un programma in linguaggio macchina.

BOOT"nome del file"[,Ddrive][<ON | ,>Udispositivo]
[,Pindirizzo di caricamento alternativo]

ESEMPI:

BOOT	carica ed esegue da un disco predisposto (per esempio CP/M).

BOOT"GRAFICA1",D0,U9	Carica il programma "GRAFICA1" dal disco 0, unità 8 e lo esegue.

BOX

Disegna un riquadro da un punto specificato dello schermo.

BOX[#colore],X1,Y1[,X2,Y2][,angolo][,riempi]

Dove:

# color	0=colore di fondo 1=colore di primo piano (default) 2=multicolore 1 3=multicolore 2
X1,Y1	Coordinate dell'angolo superiore sinistro (in scala)
X2,Y2	Coordinate diagonalmente opposte a X1,Y1 (scalate); default è l'ultima coordinata raggiunta
angolo	Rotazione in senso orario; default 0 gradi
riempi	riempie la forma di colore 0=non riempire 1=riempi default 0

ESEMPI:

BOX 1, + 10, + 10	Disegna un quadrato 10 pixel a destra ed in giù della locazione grafica corrente.
BOX 1, 10, 10, 60, 60	Disegna il contorno di un rettangolo.
BOX, 10,10,60,60,45,1	Disegna, ruota e riempie un riquadro (un diamante).
BOX, 30,90,60,90,45,1	Disegna un poligono pieno e ruotato. Ogni parametro deve essere incluso eccetto il colore, che può anche essere omesso. Strano notare come i manuali americani dicano invece che OGNI parametro può essere omesso sostituendolo con una virgola.

> **NOTA:** Se il grado di rotazione è maggiore di 360 allora i contorni si sovrappongono.

BSAVE

Salva un file binario dalle locazioni di memoria specificate.

BSAVE"nome del file" [,Ddrive [<ON | ,U>Udispositivo] [,B# banco],
Pindirizzo iniziale TO Pindirizzo finale

Dove:

- indirizzo iniziale è l'indirizzo d'inizio del file da salvare
- indirizzo finale è l'indirizzo dell'ultimo byte da salvare + 1.

Questo è simile al comando SAVE del monitor di linguaggio macchina.

ESEMPI:

BSAVE"DATI SPRITE",B0, P3584TOP4096	Salva il file binario chiamato "DATI SPRITE" dalla locazione 3584 alla 4095 nel banco 0.
BSAVE"PROG. SCR",D0, U9,B0,P3182TOP8000	Salva il file binario chiamato "PROG. SCR" da 3182 a 7999 dal banco 0, unità 9.

CATALOG

Mostra la directory del disco.

CATALOG [Ddrive][<ON| ,>Udispositivo][,stringa di ricerca]

ESEMPIO:

CATALOG Mostra la directory del drive 0, unità 8.

CHAR

Mostra dei caratteri sullo schermo alla posizione specificata.

CHAR[# colore],X,Y[,stringa][,RVS]

Questa istruzione è usata principalmente per stampare caratteri sullo schermo bit map, ma può essere usata anche sullo schermo di testo. Ecco il significato dei parametri:

# colore	0=fondo 1=primo piano
X	colonna del carattere (0-39) (VIC) (0-79) (8563)

Y riga del carattere (0-24 per entrambi gli schermi)
stringa la stringa da stampare (può essere omessa quando si desideri solo posizionare il cursore)
RVS indicatore di negativo: se 0 la stringa viene stampata normalmente, se 1 in negativo; in modo testo il negativo può essere ottenuto anche includendo un codice 18 (RVS) nella stringa

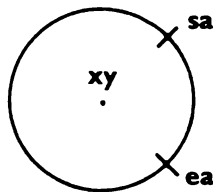
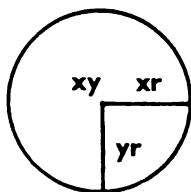
ESEMPIO:

```
10 COLOR2,3: REM MULTICOLORE 1=ROSSO
20 COLOR3,7: REM MULTICOLORE 2=BLU'
30 GRAPHIC3,1
40 CHAR0,10,10,''TESTO'',0
```

CIRCLE

Traccia cerchi, ellissi, archi, ecc. in una specifica posizione dello schermo

CIRCLE[# colore],X,Y[,rX][,rY][,ap][,af][,angolo][,inc]
dove:

# colore
0=colore di fondo
1=colore di primo piano
2=multicolore 1
3=multicolore 2
X,Y coordinate del centro
rX raggio della X (in scala); (default 0)
rY raggio della Y (in scala); (dafualt rX)
ap angolo di partenza del tracciamento (default 0 gradi)
af angolo di fine (default 360 gradi)
angolo rotazione in senso orario (default 0 gradi)
inc incremento in gradi fra ogni segmento (default 2 gradi)

ESEMPI:

CIRCLE1,160,100,65,10	Disegna un'ellisse
CIRCLE1,160,100,65,50	Disegna un cerchio (su alcuni monitor il cerchio potrà apparire come un'ellisse: ciò è dovuto al monitor e non al comando CIRCLE)

CIRCLE1,60,40,20,18,,,,45	Disegna un ottagono
CIRCLE1,260,40,20,,,,,90	Disegna un diamante
CIRCLE1,60,140,20,18,,,,120	Disegna un triangolo
CIRCLE1,+2,+2,50,50	Disegna un cerchio (due pixel a destra e in giù) relativo alle coordinate raggiunte precedentemente
CIRCLE1,30;90,	Disegna un cerchio 30 pixel orientati a 90 gradi più in là della posizione corrente.

Potete omettere un parametro, ma dovete comunque sempre piazzare una virgola al suo posto.

CLOSE

Chiude un file logico

CLOSE# di file

ESEMPIO:

CLOSE 2 Chiude il file logico #2

CLR

Cancella le variabili di un programma

CLR

CMD

Dirige l'uscita dei dati su un dispositivo diverso dal video, come il disco o la stampante.

CMD# di file [,elenco di stampa]

ESEMPIO:

OPEN1,4	Apre il dispositivo #4 (la stampante)
CMD 1	Dirige l'output verso di essa
LIST	Lista il programma su carta
PRINT#1	Ridirige l'output su video
CLOSE 1	Chiude il file logico.

COLLECT

Esegue la Validate del disco

COLLECT [Ddrive] [<ON |,> Udispositivo]

ESEMPIO:

COLLECT D0 Libera i blocchi allocati inutilmente e cancella anche i file non chiusi.

COLLISION

Definisce la riga di programma da eseguire in relazione ad un evento del video

COLLISION tipo [,riga di programma]

tipo	Tipo di interruzione, come segue: 1=collisione fra sprite 2=collisione fra sprite e sfondo 3=segnale dalla penna ottica (solo a 40 colonne)
riga di programma	Numero di linea della subroutine da eseguire (che deve infatti terminare con un'istruzione RETURN)

ESEMPI:

COLLISION 1,5000	Abilita il controllo di collisione fra sprite e, nel caso, l'esecuzione del programma da 5000 in poi
COLLISION 1	Disabilita il controllo di collisione
COLLISION 2,1000	Abilita il controllo di collisione fra sprite e fondo e, nel caso, l'esecuzione del programma dalla linea 1000.

COLOR

Definisce i colori per ogni zona dello schermo

COLOR Ncolore, colore selezionato.

Quest'istruzione assegna un colore ad una di queste sette aree di colore:

N.o	COLORE RELAZIONE sorgente
0	Fondo a 40 colonne
1	Primopiano a 40 colonne
2	Multicolore 1
3	Multicolore 2
4	Bordo a 40 colonne
5	Carattere a 40 e 80 colonne
6	Fondo a 80 colonne

I colori disponibili vanno da 1 a 16.

CODICI COLORE	COLORE	CODICI COLORE	COLORE
1	**Nero**	**9**	**Arancione**
2	**Bianco**	**10**	**Marrone**
3	**Rosso**	**11**	**Rosso chiaro**
4	**Azzurrino**	**12**	**Grigio scuro**
5	**Porpora**	**13**	**Grigio medio**
6	**Verde**	**14**	**Verde chiaro**
7	**Blù**	**15**	**Azzurro**
8	**Giallo**	**16**	**Grigio chiaro**

Numeri dei colori a 40 colonne

CODICI COLORE	COLORE	CODICI COLORE	COLORE
1	**Nero**	**9**	**Porpora scuro**
2	**Bianco**	**10**	**Giallo scuro**
3	**Rosso scuro**	**11**	**Rosso chiaro**
4	**Azzurro chiaro**	**12**	**Azzurro scuro**
5	**Porpora chiaro**	**13**	**Grigio medio**
6	**Verde Scuro**	**14**	**Verde chiaro**
7	**Blù scuro**	**15**	**Azzurro**
8	**Giallo chiaro**	**16**	**Grigio chiaro**

Numeri dei colori a 80 colonne

ESEMPI:

COLOR 0,1: Fondo a 40 colonne in nero
COLOR 5,8: Carattere in giallo.

CONCAT

Concatena due file sequenziali

CONCAT" file 2" [,Ddrive] TO"file 1"
[,Ddrive][<ON |,>Udispositivo]

ESEMPI:

CONCAT"file B"TO"file A"	Crea un file di nome A formato dal file B unito al precedente file A, che viene così perso.
CONCAT(A$)TO(B$),D1,U9	Crea un file col nome contenuto in B$ formato dai file coi nomi contenuti in A$ e B$, sul disco 1, unità 9 (su disco doppio).

Quando si usa una variabile come nome di file, quest'ultima deve essere racchiusa fra parentesi.

CONT

Riprende l'esecuzione del programma interrotto da STOP, END e dal tasto RUN/STOP

CONT

COPY

Copia un file da un disco ad un altro con un'unità doppia oppure con una singola ma con un nome diverso

COPY [Ddrive,]"nome sorgente"TO[Ddrive,]
"nome destinazione"[<ON |,>Udispositivo]

> **NOTA:** Non è possibile effettuare copie fra unità disgiunte, sia singole che doppie; il comando corrisponde al comando COPY del DOS del drive.

ESEMPI:

COPYD0,"TEST"TOD1,"PROG TEST"	Copia TEST dal drive 0 al drive 1 col nome "PROG TEST"
COPYD0,"PIENO"TOD1,"PIENO"	Copia "PIENO" dal drive al drive 1
COPY D0 TO D1	Copia tutti i file dal drive 0 al drive 1
COPY"PROG.LAV"TO"BACKUP"	Copia "PROG.LAV" come "BACKUP" sullo stesso drive (drive 0).

DATA

Identifica dei dati da usare nel programma

DATA lista di costanti

ESEMPIO:

DATA 100,200,FRED,"CIAO, MAMMA",,3,14,ABC123

DCLEAR

Chiude tutti i canali aperti verso il disco ed inizializza il drive

DCLEAR [Ddrive] [<ON |,>Udispositivo]

ESEMPI:

DCLEAR D0	Chiude tutti i canali verso il drive 0, unità 8 e lo inizializza
DCLEAR D1,U9	Chiude tutti i canali verso il drive 1, unità 9 e lo inizializza.

DCLOSE

DCLOSE [# file] [<ON |,> Udispositivo]

Chiude un file su disco senza inizializzare l'unità

ESEMPI:

DCLOSE	Chiude tutti i file aperti verso l'unità 8
DCLOSE#2	Chiude tutti i file aperti verso l'unità 9.

DEF FN

Definisce una funzione dell'utente

DEF FN nome(variabile)=espressione

ESEMPIO:

```
10 DEF FNA(X)=12*(34.75-X/.3)+X
20 PRINT FNA(7)
```

Il numero 7 viene inserito al posto della X nella formula data nella definizione di funzione. Nell'esempio sopra viene ritornato il valore 144.

NOTA: Se pensate di usare una funzione da voi definita in un programma che fa anche uso di comandi grafici fate attenzione a definire la funzione dopo il comando GRAPHIC, poichè esso distrugge l'area da 7168 a 16383, dove vengono anche memorizzate le definizioni di funzione.

DELETE

Cancella le linee di programma specificate

DELETE [linea iniziale][-linea finale]

ESEMPI:

DELETE 75	Cancella la linea 75
DELETE 10-50	Cancella le linee da 10 a 50 comprese
DELETE-50	Cancella fino a 50 compresa
DELETE75-	Cancella da 75 in poi

DIM

Dichiara il numero di elementi di un vettore

DIM variabile (indice) [,variabile(indice)]...

ESEMPIO:

10 DIMA$(40),B7(15),CC%(4,4,4)

Dimensiona tre vettori in cui A$, B7 e CC% hanno rispettivamente 41, 16 e 125 elementi.

DIRECTORY

Mostra sullo schermo il contenuto di un disco

DIRECTORY [Ddrive][<ON |,>Udispositivo][,stringa di ricerca]

ESEMPI:

DIRECTORY	Lista tutti i file del disco nel drive 0, unità 8
DIRECTORY D1,U9,"LAVORO"	Mostra il file chiamato "LAVORO" nel drive 1, unità 8
DIRECTORY"AB*"	Lista tutti i file il cui nome inizia per AB; l'asterisco (*) equivale a tutti i caratteri seguenti AB.
DIRECTORY D0,"?.BAK"	Lista tutti i file di cinque lettere che finiscono con ".BAK"; il punto interrogativo (?) sostituisce una qualsiasi lettera

DIRECTORY D1,U9,(A$)	Lista tutti i file corrispondenti alla stringa contenuta nella variabile A$; ricordate di racchiudere sempre i nomi di variabili fra parentesi.

> **NOTA:** Se volete stampare su carta la DIRECTORY di un disco seguite questa procedura:
> LOAD"$0",8
> OPEN4,4: CMD4:LIST
> PRINT#4:CLOSE4

DLOAD

Carica un programma BASIC dal drive 0, unità 8

DLOAD"nome del file"[,Ddrive] [<ON |,>Udispositivo]

ESEMPI:

DLOAD"BANKRECS"	Carica, se presente sul disco, il programma "BANKRECS"
DLOAD(A$)	Carica un programma il cui nome è contenuto nella variabile A$; se la variabile è nulla si avrà un messaggio d'errore.

DO/LOOP/WHILE/UNTIL/EXIT

Definiscono e controllano un loop (anello, giro) di programma

DO [UNTIL condizione | WHILE condizione]
istruzioni[EXIT]
LOOP [UNTIL condizione | WHILE condizione]

Questa struttura ad anello esegue le istruzioni fra DO e LOOP. Se nè UNTIL nè WHILE accompagnano le istruzioni DO o LOOP la struttura verrà ripetuta all'infinito. Allora l'unico modo di uscire da DO/LOOP sarà l'istruzione EXIT, che quando incontrata trasferisce il controllo all'istruzione successiva LOOP. Gli anelli DO/LOOP possono essere nidificati, esattamente come un ciclo FOR/NEXT. Se si specifica una condizione UNTIL l'esecuzione continuerà fino al momento in cui la condizione specificata diventerà vera: allora si uscirà dall'anello. Se invece si specifica la condizione WHILE l'esecuzione continuerà mentre la condizione sarà vera, uscendo dall'anello appena la condizione sarà falsa. Un'esempio di condizione può essere A=1 o G>65.

ESEMPI:

```
10 x = 25
20 DO UNTIL X = 0
30 X = X-1
40 PRINT"X=";X
50 LOOP
60 PRINT"fine dell'anello"
```

Questo esempio esegue le istruzioni X=X-1 e PRINT"X=";X finchè X=0. Allora uscirà dal loop e stamperà "fine dell'anello"

```
10 DO WHILE A$<>CHR$(13):GETKEY A$:PRINTA$:LOOP
20 PRINT"HAI PREMUTO IL TASTO «RETURN»"
```

Questa struttura DO aspetta la pressione di un tasto, stampa il carattere corrispondente e se corrisponde al tasto «RETURN» allora esce dall'anello ed esegue la linea 20

```
10 DOPEN#8,"FILESEQ"
20 DO
30 GET#8,A$
40 PRINTA$;
50 LOOP UNTIL ST
60 DCLOSE#8
```

Questo programma apre un file sequenziale e lo legge fino alla fine, indicata dalla variabile ST

DOPEN

Apre un file su disco per operazioni di lettura/scrittura

DOPEN# di file,"nome del file[,<tipo>]"[,Llunghezza del record] [,Ddrive][<ON ,>Udispositivo][,W]

dove il tipo può essere:
- **S** = file sequenziale
- **P** = file programma
- **U** = file utente
- **L** = lunghezza del record, se si apre un file relativo
- **W** = file aperto per la scrittura, se di tipo S, P o U

ESEMPI:

DOPEN#1,"INDIRIZZO",W — Apre il file sequenziale INDIRIZZO per la scrittura

DOPEN#2,"RICETTA",D1,U9 — Apre il file sequenziale RICETTA sul disco 1, unità 9 per leggerlo.

DRAW

Traccia punti, linee e forme in una posizione dello schermo grafico

DRAW [N.o colore][,X1,Y1][TOX2,Y2]...

dove:

N.o colore	0=Fondo bit map 1=Primo piano bit map 2=Multicolore 1 3=multicolore 2
X1,Y1	coordinate d'inizio (0,0-319,199)
X2,Y2	coordinate di fine (0,0-319,199)

ESEMPI:

DRAW1,100,50	Traccia un punto
DRAW,10,10 TO 100, 60	Traccia una linea
DRAW,10,10 TO 10, 60 TO 100, 60 TO 10, 10	Traccia un triangolo
DRAW 1,120; 45	Traccia un punto distante 120 pixel a 45 gradi dalla posizione corrente
DRAW	Traccia un punto alla posizione attuale. Per posizionarvi senza tracciare usate LOCATE.

Ogni parametro può essere omesso ma dovete comunque specificare la virgola relativa; i parametri omessi assumeranno i valori di default.

DSAVE

Salva un programma BASIC su disco

DSAVE"nome del file"[,Ddrive][<ON, |>Udispositivo]

ESEMPI:

DSAVE"BANKRECS"	Salva il programma "BANKRECS" su disco
DSAVE(A$)	Salva il programma col nome contenuto in A$
DSAVE"PROG 3",D1,U9	Salva il programma "PROG 3" sul disco 1, unità 9.

DVERIFY

Verifica il programma in memoria con quello su disco

DVERIFY"nome del file"[,Ddrive] [<ON |,>Udispositivo]

Per verificare file **binari,** date un'occhiata a VERIFY"nome del file",8,1 nella descrizione del comando VERIFY

ESEMPI:

DVERIFY"C128"	Verifica il programma "C128" sul drive 0, unità 8
DVERIFY"SPRITE",D0,U9	Verifica il programma "SPRITE" sul drive 1, unità 9.

END

Termina l'esecuzione del programma

END

ENVELOPE

Definisce l'inviluppo di uno strumento musicale

ENVELOPE n[,att] [,dec] [,sos] [,ril] [,fdo] [,pul]

dove:

n	Numero dell'inviluppo (0-9)
att	Velocità di attacco (0-15)
dec	Velocità di decadimento (0-15)
sos	Livello di sostegno (0-15)
ril	Velocità di rilascio (0-15)
fdo	Forma d'onda: 0=triangolare 1=dente di sega 2=pulsazione (onda quadra) 3=rumore 4=modulazione ad anello
pul	Ampiezza della pulsazione (0-4095)

Guardate l'opzione T del comando PLAY per selezionare l'inviluppo della stringa di note.

ESEMPIO:

ENVELOPE 1, 10, 5, 10, 0, 2,2048	Questo comando pone l'inviluppo 1 a: attacco=10; decadimento=5; sostegno=10; rilascio=0; forma d'onda pulsazione e ampiezza a 2048.

FAST

Fa girare il processore 8502 a 2MHz

FAST

Questo comando seleziona il modo di funzionamento a 2MHz, disabilitando lo schermo a 40 colonne. Tutte le operazioni sono notevolmente accelerate. La grafica può essere usata, ma non sarà visibile fino ad un comando SLOW. Il Commodore 128 si predispone, all'accensione, a 1MHz. Le operazioni in DMA (FETCH, SWAP e STASH) devono essere effettuate ad 1MHz.

FETCH

Preleva dei dati dal modulo di espansione di memoria

FETCH# di byte,indint,indest,estban

dove:

byte	numero di byte da prelevare dall'espansione (0-65535) con 0=64K (65536)
indint	indirizzo d'inizio della memoria centrale, dove memorizzare i byte
indest	indirizzo d'inizio della memoria d'espansione, da dove prelevare i byte
estban	banco dell'espansione, da 0 a 7

Il banco di memoria centrale viene selezionato tramite il comando BANK. Il banco di RAM in DMA per il chip VIC II viene selezionato dai bit 6-7 del registro di configurazione RAM in 54534 ($D506).

FILTER

Definisce i parametri del filtro del suono del chip SID

FILTER [freq] [,pb] [,pbn] [,pa] [,ris]

dove:

freq	frequenza di taglio del filtro
pb	filtro passa-basso acceso/spento (1/0)
pbn	filtro passa-banda acceso spento(1/0)
pa	filtro passa-alto acceso/spento (1/0)
ris	risonanza (0-15)

L'omissione di un parametro lo lascerà invariato

ESEMPI:

FILTER 1024, 0, 1, 0, 2	Pone il taglio a 1024, seleziona il passabanda con una risonanza di 2
FILTER 2000, 1, 0, 1, 10	Pone il taglio a 2000, seleziona i filtri passaalto e basso con una risonanza di 10.

FOR/TO/NEXT/STEP

Definisce una struttura ripetitiva

FOR variabile=valore iniziale TO valore finale [STEP incremento]
NEXT [variabile]

La logica dell'istruzione FOR/NEXT è la seguente: la variabile di FOR è posta al valore iniziale; quando il programma raggiunge l'istruzione NEXT l'incremento specificato da STEP è aggiunto alla variabile di FOR; se questa variabile è maggiore del valore finale allora il programma prosegue dopo il NEXT; altrimenti il ciclo viene eseguito ancora. E' vero il contrario se l'incremento è negativo.

ESEMPIO:

```
10 FORL=1 TO 10
20 PRINT L
30 NEXT L
40 PRINT "HO FINITO! L = "L
```

Questo programma stampa i numeri da 1 a 10 seguiti dal messaggio HO FINITO! L= 11

ESEMPIO:

```
10 FOR L=1 TO 100
20 FOR A=5 TO 11 STEP.5
30 NEXT A
40 NEXT L
```

È anche possibile specificare più variabili per ogni NEXT: qui è equivalente fare NEXTA,L al posto di NEXTA:NEXTL. Il loop FOR/NEXT delle linee 20 e 30 è nidificato in quello delle linee 10 e 40. L'incremento di .5 dimostra che è possibile usare incrementi reali. Il loop più interno deve giacere completamente dentro il loop più esterno (linee 10 e 40).

GET

Riceve dati in input dalla tastiera, uno alla volta, senza aspettare la pressione di un tasto. Nel caso non vi sia niente nel buffer di tastiera viene ritornato un carattere nullo.

GET lista di variabili

ESEMPIO:

10 DO:GETA$:LOOP UNTIL A$="A"	Questa linea aspetta la pressione del tasto A

20 GET B,C,D Prende 3 valori dalla tastiera anche senza premere tasti

GETKEY

Riceve dati in input dalla tastiera, uno alla volta, aspettando la pressione di un tasto.

GETKEY lista di variabili

ESEMPI:

10 GETKEY A$
Questa linea aspetta la pressione di un tasto.

10 GETKEY A$,B$,C$
Questa linea aspetta la pressione di tre tasti.

GET#

Riceve dati in input da tastiera, nastro, disco e RS232, uno alla volta.

GET# numero di file, lista di variabili

ESEMPI:

10 GET # 1,A$ Questa linea riceve un carattere dal file #1, che deve essere stato precedentemente aperto. Anche se è possibile aprire un file verso la tastiera, ciò è inutile data la presenza dell'istruzione GET che riceve già da tastiera.

GO64

Passa in modo C64.

GO64

Per ritornare in modo 128 premete il pulsante di reset o spegnete ed accendete il computer.

GOSUB

Chiama la subroutine col numero di linea specificato.

GOSUB # di linea

ESEMPI:

20 GOSUB 800

800PRINT "IL C128 E' VALSO L'ATTESA!":RETURN

GOTO/GO TO

Prosegue l'esecuzione del programma dal numero di linea specificato.

GOTO N.o di linea

ESEMPIO:

10 PRINT"COMMODORE" 20 GOTO 10	Il GOTO della linea 20 fa in modo che la linea 10 sia ripetuta fino alla pressione di RUN/STOP.
GOTO 100	Inizia l'esecuzione del programma dalla linea 100, senza cancellare le variabili (al contrario di RUN 100)

GRAPHIC

Seleziona un modo grafico.

1) GRAPHIC modo [,cancella] [,riga]
2) GRAPHIC CLR

dove:
modo, indica un modo grafico; cancella, cancella (1) o no (0) lo schermo; riga, riga di partenza della finestra di testo (0-24); CLR disalloca e disabilita la pagina grafica (se presente, deve essere l'unico parametro). Questa istruzione pone il Commodore 128 in uno di questi sei modi grafici:

MODO	DESCRIZIONE
0	testo a 40 colonne (default)
1	pagina grafica (bit map)
2	bit map e finestra di testo
3	bit map multicolore
4	bit map multicolore e finestra di testo
5	testo a 80 colonne

ESEMPI:

GRAPHIC 1,1	Seleziona e cancella la bit map
GRAPHIC 4, 0, 10	Seleziona la bit map multicolore senza cancellarla, con una finestra di testo dalla riga 10 a 24
GRAPHIC 0	Disabilita la bit map senza disallocarla
GRAPHIC 5	Seleziona il testo a 80 colonne
GRAPHIC CLR	Disabilita e disalloca la bit map senza cancellarla

GSHAPE

Vedere SSHAPE.

HEADER

Formatta un dischetto

HEADER"nome del disco "[,Iid] [,Ddrive]
[<ON |,> Udispositivo]

Prima di usare per la prima volta un dischetto bisogna formattarlo tramite il comando HEADER. Questo comando si può anche usare per riformattare un dischetto e riusarlo.

Quando immettete il comando HEADER in modo diretto e premete «RETURN», appare il messaggio **ARE YOU SURE?** (SEI SICURO?), mentre non appare in modo programma. Rispondete premendo Y e «RETURN» per confermare, altrimenti qualsiasi altro tasto. Il comando HEADER è analogo al comando del BASIC 2.0.

OPEN 1, 8, 15, "N0:nome del disco,id":CLOSE 1

ESEMPI:

HEADER "DISCOMIO",I23, D0	Formatta il disco con "DISCOMIO" e id 23 sul drive 0, unità 8.
HEADER "RECS",I45, D1 ON U9	Formatta il disco con "RECS" e id 45 sul drive 1, unità 9
HEADER "PROGRAMMI C128",D0	Effettua una formattazione veloce sul drive 0, unità 8, assumendo che il disco sia già formattato (cancella solo la directory)
HEADER (A$), I76, D0, U9	Formatta il disco col nome in A$ e id 76 sul drive 0, unità 9. Il codice id NON può essere specificato tramite una variabile.

HELP

Evidenzia la linea che ha originato l'errore.

HELP

Il comando HELP è usato dopo una segnalazione d'errore in un programma. Sullo schermo a 40 colonne appare la linea errata con l'errore in negativo; sullo schermo a 80 l'errore è sottolineato.

IF/THEN/ELSE

Valuta un'espressione condizionale ed esegue determinate parti di un programma a seconda del risultato dell'espressione.

IF espressione THEN istruzioni [:ELSE clausola else]

L'istruzione IF...THEN valuta un'espressione BASIC e sceglie una fra due vie d'esecuzione del programma in relazione al risultato dell'espressione. Se l'espressione è vera (-1) verrà eseguita la parte seguente THEN, che può essere sia una o più istruzioni, che un numero di linea da cui riprendere l'esecuzione. Se invece l'espressione è falsa (0) allora l'esecuzione verrà ripresa dalla linea seguente, a meno che non sia presente una clausola ELSE. In ogni caso la linea contenente l'IF...THEN non può essere più lunga di 160 caratteri. Guardate anche BEGIN/BEND. La clausola ELSE, se presente, deve essere sulla stessa linea di IF...THEN e separata da due punti (:).

La clausola ELSE viene eseguita solo se la condizione IF è falsa. L'espressione da valutare può essere una variabile od una formula,e può anche includere gli operatori relazionali (=,<,>,<=,>=,<>).

ESEMPI:

```
50 IF X>0 THEN PRINT "OK":ELSE END
```

Questa linea controlla se il valore di X è maggiore di zero: se così allora stampa "OK" ed ignora la clausola ELSE. Altrimenti non esegue l'istruzione PRINT e passa ad eseguire l'istruzione dopo ELSE.

```
10 IF X = 10 THEN 100
20 PRINT "X NON E' UGUALE
A 10"
:
99 STOP
100 PRINT "X E' UGUALE A 10"
```

Questo programma controlla se X=10: se è così salta alla linea 100 e stampa "X E' UGUALE A 10", altrimenti stampa "X NON E' UGUALE A 10" e continua fino alla linea 99, dove si ferma (STOP).

INPUT

Riceve una stringa di dati od un numero dalla tastiera aspettando la pressione del tasto «RETURN».

INPUT ["stringa di avvertimento";] lista di variabili

ESEMPIO:

```
10 INPUT "PER FAVORE SCRIVI UN NUMERO";A
20 INPUT "ED IL TUO NOME";A$
30 PRINT A$" HAI SCRITTO IL NUMERO";A
```

INPUT#

Riceve dati da un canale di I/O ponendoli in una variabile numerica o stringa.

INPUT# file, lista di variabili

ESEMPIO:

```
10 OPEN 2,8,2
20 INPUT# 2, A$, C, D$
```

Questa istruzione riceve i dati memorizzati in A$, C e D$ dal file disco #2 aperto nella linea 10.

KEY

Definisce od elenca le definizioni dei tasti funzione.

KEY [numero tasto, stringa da assegnare]

La lunghezza complessiva di tutte le definizioni è di 241 caratteri.

ESEMPI:

```
KEY 7,"GRAPHIC0" + CHR$(13) + "LIST" + CHR$(13)
```

Quando premete, in modo diretto, il tasto F7 il computer selezionerà lo schermo di testo a 40 colonne e listerà il programma. CHR$(13) è il codice ASCII corrispondente al tasto «RETURN». Per il tasto ESCape usate CHR$(27), mentre per le virgolette (") CHR$(34). Potete ridefinire i tasti anche da programma. Per esempio:

```
10 KEY2, "PRINT DS$" + CHR$(13)
```

Questo fa in modo che il computer controlli il canale degli errori del drive ogni volta che premete F2.

```
10 FOR I=1 TO 7 STEP 2
20 KEY I, CHR$(I + 132):NEXT
30 FOR I=2 TO 8 STEP 2
40 KEY I, CHR$(I + 132):NEXT
```

Questo definisce i tasti funzione come immagine di sè stessi.

LET

Assegna un valore ad una variabile.

[LET] variabile = espressione

ESEMPIO:

```
10 LET A = 5                    Rende A uguale a 5
```

20 B = 6	Rende B uguale a 6
30 C = A * B + 3	Rende C uguale a 33
40 D$ = "CIAO"	Rende D$ uguale a "CIAO"

LIST

Lista il programma BASIC in memoria.

LIST [linea iniziale] [-linea finale]

In Modo C128 LIST può essere usato anche da programma senza interromperne l'esecuzione.

ESEMPI:

LIST	Mostra tutto il programma
LIST 100-	Mostra le linee da 100 in poi
LIST 10	Mostra la linea 10
LIST-100	Mostra le linee fino a 100
LIST 10-200	Mostra le linee da 10 a 200

LOAD

Carica un programma da un dispositivo periferico quale registratore o drive.

LOAD "nome del file" [,dispositivo] [,flag di rilocazione]

Questo è il comando usato per caricare un programma da cassetta o disco. Qui, il nome del file ha una lunghezza massima di 16 caratteri ed è compreso fra virgolette. Dopo le virgolette può esserci una virgola seguita dal numero di dispositivo (1=cassetta;8-11=disco). Se omettete il dispositivo il C128 procederà al caricamento da registratore. Dopo il dispositivo viene il flag di rilocazione: questo indica se caricare il programma nella zona BASIC (0) oppure nella zona originaria (1), che peraltro può benissimo essere la zona BASIC.

ESEMPI:

LOAD	Legge il prossimo programma su nastro
LOAD "CIAO"	Ricerca sul nastro il programma CIAO e, se trovato, lo carica
LOAD(A$),8	Carica da disco il programma il cui nome è in A$. Qui A$, trattandosi di un comando compatibile col BASIC 2.0, avrebbe anche potuto non essere racchiuso fra parentesi
LOAD"CIAO",8	Cerca il programma chiamato CIAO sul drive 0, dispositivo 8 (equivalentea DLOAD"CIAO")
LOAD"LINGMAC",8,1	Carica il programma LINGMAC nelle locazioni originarie da cui era stato salvato.

LOCATE

Posiziona il cursore pixel (cioè la posizione di tracciamento) a specifiche coordinate della pagina grafica.

LOCATE X,Y

ESEMPI:

LOCATE 160,100	Posiziona il CP (cursore pixel) al centro dello schermo senza tracciare nulla
LOCATE + 20,100	Muove il CP 20 pixel a destra della posizione X all'altezza Y di 100
LOCATE 30;270	Muove il CP 30 pixel a sinistra mantenendosi sulla stessa Y.

NOTA: l'ultimo esempio differisce da quello del manuale americano; là erano infatti usate coordinate relative negative (LOCATE-30,+20) che sui modelli europei danno un errore. D'altra parte là non si fa menzione del posizionamento tramite coordinate angolari, come invece qui riportato: questo posizionamento si effettua dando uno spostamento di X pixel in direzione di un angolo Y da 0 a 360 separati da un punto e virgola (;). 270 corrisponde infatti ad uno spostamento orizzontale a sinistra. Per conoscere la posizione del CP usate la funzione RDOT(x), con x da0-2 per ottenere rispettivamente la coordinata X, la Y ed il colore del punto corrente.

MONITOR

Entra nel monitor di linguaggio macchina del C128.

MONITOR

Andare nella III parte per avere maggiori dettagli.

MOVSPR

Posiziona o muove lo sprite sullo schermo.

1) MOVSPRnumero,X,Y	Piazza lo sprite alle coordinate specificate
2) MOVSPRnumero,+/-X,+/-Y	Piazza lo sprite a tanti pixel X e Y dalla posizione corrente

3) MOVSPRnumero,X;Y Piazza lo sprite secondo le coordinate angolari; lo sposta di X pixel in direzione dell'angolo Y da 0 a 360

4) MOVSPR numero,angolo # velocità Inizia il movimento dello sprite secondo l'angolo X (0-360) alla velocità Y (0-15).

Queste istruzioni muovono uno sprite (da 1a 8) secondo le coordinate di sprite, che vanno da 24,50 a 343,249, invece che da 0,0 a 319,199. Il sistema di coordinate di sprite è descritto nella IV parte.

ESEMPI:

MOVSPR 1,150,150	Posiziona lo sprite #1 vicino al centro dello schermo
MOVSPR 1,+20,-30	Muove lo sprite # 1 20 pixel a destra e 30 in alto della posizione corrente
MOVSPR 4,50;45	Muove lo sprite #4 di 50 pixel verso un angolo di 45 gradi
MOVSPR 5,45 #15	Inizia il movimento dello sprite #5 con un angolo di 45 gradi ad una velocità di 15.

NOTA: una volta iniziato il movimento di uno sprite, come nel quarto esempio, esso continuerà anche a sprite disabilitato, finchè non si premerà RUN/STOP/ RESTORE oppure si darà MOVSPR numero,0#0. Ancora, il comando SCALE influenza anche le coordinate di sprite, che dovranno quindi essere adeguate.

NEW

NEW Cancella il programma BASIC assieme alle variabili (il programma non viene comunque cancellato realmenteed è ancora possibile recuperarlo).

ON

Esegue un salto ad un certo numero di linea in relazione al risultato di un'espressione.

ON espressione <GOTO|GOSUB> linea #1 [,linea #2,...]

ESEMPIO:

```
10 INPUT X:IFX <0 THEN 10
20 ON X GOTO 30,40,50,60
25 STOP
30 PRINT" X = 1"
40 PRINT" X = 2"
50 PRINT" X = 3"
60 PRINT" X = 4"
```

Come si può vedere, l'espressione deve dare un valore da 1 a n; per valori come 0 o superiori al numero delle linee specificate il programma prosegue dopo l'istruzione ON. In questo caso se X=0 o X>4 il programma incontra l'istruzione STOP.

OPEN

Apre i file per l'input/output

OPEN # di file, dispositivo [,indirizzo secondario]
<[,"nome del file [,tipo] [,[modo"]] |<", stringa di comando>]

ESEMPI:

10 OPEN 3,3	Apre lo schermo come file # 3
20 OPEN 1,0	Apre la tastiera come file #1
30 OPEN 1,1,0,"PUNTO"	Apre il registratore come file #1 per la lettura del file PUNTO
OPEN 4,4	Apre la stampante come file # 4
OPEN 15,8,15	Apre il canale 15 (canale di comando) del disco come file#15
5 OPEN 8,8,12,"FILETEST,S,W"	Apre un file sequenziale su disco per la scrittura. Guardate anche le istruzioni CLOSE, CMD, GET#, INPUT# e PRINT# e le variabili di sistema ST, DS e DS$.

PAINT

Riempie un'area grafica con un colore

PAINT [sorg. colore] ,X,Y [,modo]

dove:

sorg. colore	0=fondo del disegno 1=primo piano 2=multicolore 1 3=multicolore 2
X,Y	coordinate di partenza (in scala)
modo	0=riempie l'area circondata dal colore sorgente scelto 1=riempie qualsiasi area di colore sorg. 0 (cioè fondo, aree vuote) circondata da qualsiasi altro colore.

Il comando PAINT riempie un'area con un determinato colore. Riempie tutta l'area partendo dalle coordinate specificate fino a trovare un margine dello stesso colore sorgente indicato nell'istruzione. Per esempio, se disegnate un cerchio col colore di primo piano (1), cominciate il riempimento dove le coordinate corrispondono al colore di fondo. Il Commodore 128 riempirà soltanto dove il colore sorgente delle coordinate X,Y differisce da quello specificato nell'istruzione PAINT. Le coordinate iniziali devono naturalmente essere all'interno della forma da riempire, e il colore sorgente di PAINT deve differire da quello corrispondente a queste coordinate.

ESEMPI:

10 CIRCLE 1,160,100,65,50	Traccia una circonferenza
20 PAINT 1,160,100	Riempie il cerchio col colore sorgente 1 (primo piano), a condizione che il punto a 160,100 sia diverso da questo colore
10 BOX 1,10,10,20,20	Traccia un quadrato
20 PAINT 1,15,15	Riempie il quadrato col colore N.1
30 PAINT 1,+10,+10	Riempie lo schermo col colore di primo piano partendo da 10 pixel a destra e in basso dalla coordinata corrente

100 PAINT 1,100;90	Riempie l'area 100 pixel in direzione 90 gradi dalla posizione corrente.

PLAY

Definisce e suona note ed elementi musicali in una stringa

PLAY "Vn,On,Tn,Un,Xn,elementi,note"
dove:

Vn = Voce (n=1-3)
On = Ottava (n=0-6)
Tn = Inviluppo (n=0-9):
0=pianoforte
1=fisarmonica
2=calliope
3=tamburo
4=flauto
5=chitarra
6=clavicembalo
7=organo
8=tromba
9=xilofono
Un = Volume (n=0-9)
Xn = Filtro acceso (n=1), spento (n=0)
note = A, B, C, D, E, F, G LA SI DO RE MI FA SOL
elementi = # .... diesis
$ .... bemolle
W .... intero
H .... metà
Q .... quarto
I .... ottavo
S .... sedicesimo
. ... nota puntata
R .... pausa
M .... attende che tutte le voci abbiano eseguito la misura corrente

L'istruzione PLAY permette di selezionare la voce, l'ottava e l'inviluppo (fra i dieci predefiniti e gli altri definiti dall'utente), il volume, il filtro e le note da suonare. Tutti i controlli vanno scritti fra virgolette. Si possono anche includere degli spazi per una maggior leggibilità senza influenzare il suono.

Tutti gli elementi eccetto R e M precedono le note a cui si riferiscono.

ESEMPI:

PLAY"V1O4T0U5X0CDEFGAB"	Suona la scala musicale (DO RE MI...) con la voce 1, ottava 4, inviluppo 0 (pianoforte) e il filtro spento
PLAY"V3O5T6U7X1#B$AW.CHDQEIF"	Suona SI diesis, LA bemolle, DO intero puntato, RE metà, MI quarto, FA ottavo.
A$="V3O5T6U3ABCDE":PLAY A$	Suona le note e gli elementi nella stringa A$
PLAY"V1CV2EV3G"	Suona un accordo con gli elementi di default.

POKE

Cambia i contenuti di una locazione di memoria RAM:

POKE indirizzo, valore

ESEMPIO:

10 POKE 53280,1 Cambia il colore del bordo a 40 colonne.

PRINT

Stampa sullo schermo di testo

PRINT [lista di stampa]

La parola PRINT può essere seguita da:

Caratteri fra virgolette	("testo")
Nomi di variabili	(A, B, A$, X$)
Funzioni	(SIN(23), ABS(33))
Espressioni	(2+2, A+3, A=B)
Segni di punteggiatura	(;,)

ESEMPIO:	**RISULTATI**
10 PRINT"CIAO"	CIAO
20 A$ = "LA'":PRINT"CIAO"A$	CIAO LA'
30 A = 4:B=2:?A+B	6
40 J = 41:PRINTJ;:PRINTJ-1	41 40
50 PRINTA;B;:D = A+B:PRINTD;A-B	4 2 6 2

Guardate anche le funzioni POS, SPC, TAB e CHAR.

PRINT#

Invia dati ai file aperti.

PRINT# numero di file [,lista di stampa]

PRINT# è seguito da un numero che si riferisce al file aperto precedentemente.

ESEMPI:

10 OPEN 4,4 20 PRINT# 4,"CIAO LA'!",A$,B$	Manda CIAO LA'! ed il contenuto di A$ e B$ alla stampante
10 OPEN 2,8,2 20 PRINT# 2,A,B$,C,D	Manda al disco i dati contenuti nelle variabili.

NOTA: il comando PRINT# è usato anche da solo per chiudere il canale dopo aver dato uncomando CMD, come segue:

```
10 OPEN 4,4
20 CMD 4:PRINT"STAMPA PAROLE"
30 PRINT#4
50 CLOSE4
```

PRINT USING

Stampa usando un formato predefinito

PRINT [# di file,] USING"lista di formato";lista di stampa

Questo comando definisce il formato di stampa di una stringa o un numero indirizzandolo verso il dispositivo selezionato.

ESEMPIO:

```
5 X = 32:Y = 100.23:A$ = "128"
10 PRINT USING"$##.### ";13.25,X,Y
20 PRINT USING"###>#";"CBM",A$
```

Quando lo eseguite, la linea 10 stampa:

$13.25 $32.00 $*****

- Y non viene stampata perchè piùlunga del formato; al suo posto appaiono cinque asterischi. La linea 20 stampa invece: CBM 128 CBM e 128 sono allineati a destra del formato.

Il segno # riserva uno spazio nel formato: quindi, se il dato da stampare è più lungo dei segni # presenti nel formato, al suo posto verranno stampati tanti asterischi quanti sono i segni #.

ESEMPIO:

```
10 PRINT USING "####";X
```

Per i seguenti valori di X viene stampato:

```
X = 12.34        12
X = 567.89      568
X = 123456     ****
```

Nella stampa di una stringa, questa viene troncata al margine del campo, stampando così solo tanti caratteri quanti sono i segni #.

ESEMPI:

CAMPO	ESPRESSIONE	RISULTATO	COMMENT
##.#	-.1	-0.1	aggiunto uno zero in testa
##.#	1	1.0	aggiunto uno zero in coda
####	-100.5	-101	arrotondato
####	-1000	****	più di quattro posti
###.	10	10.	aggiunto il punto decimale
#$##	1	$1	segno mobile del dollaro

PUDEF

Ridefinisce i simboli usati nell'istruzione PRINT USING

PUDEF "nnnn"

Dove "nnnn" è ogni combinazione di caratteri fino ad un massimo di quattro. PUDEF permette di ridefinire ognuno dei seguenti quattro simboli: spazi, virgole, punti decimali e segni di dollaro. Questi simboli possono essere cambiati con altri sostituendoli nella corretta posizione. La posizione 1 è il carattere di riempimento, normalmente uno spazio. Piazzate qui ogni carattere vogliate al posto dello spazio. La posizione 2 è la virgola, e stampa una virgola. La posizione 3 è il punto decimale, e stampa un punto decimale. La posizione 4 infine è il dollaro, e stampa un dollaro. Una volta ridefinite le posizioni, per far stampare i vostri segni dovrete comunque mettere, nella stringa di PRINT USING, i segni di default, cioè # , . $: in corrispondenza di questi segni verrà stampato il vostro carattere.

ESEMPI:

10 PUDEF "*"	stampa un asterisco al posto dello spazio
20 PUDEF" <"	stampa un < al posto della virgola.

READ

Legge dei dati da un'istruzione DATA e li pone in una variabile

READ lista di variabili

In un programma potete leggere i dati e poi rileggerli usando l'istruzione RESTORE. Questa istruzione pone il puntatore ai dati alla linea col numero specificato nell'istruzione RESTORE oppure, se non specificato, alla prima linea DATA del programma. Guardate anche le istruzioni DATA e RESTORE.

ESEMPI:

10 READ A,B,C 20 DATA 3,4,5	Legge i primi tre valori numerici dalla più vicina linea DATA
10 READ A$,B$,C$ 20 DATA JOHN, PAUL, GEORGE	Legge i primi tre valori stringa dalla più vicina linea DATA
10 READ A,B$,C 20 DATA 1200, NANCY,345	Legge una variabile numerica, una stringa ed ancora una numerica.

RECORD

Posiziona il puntatore al record di un file relativo

RECORD # numero di file, # di record [,byte]

Quando vi posizionate su un record inesistente può succedere: se eseguite un PRINT#, allora verranno creati tutti i record dall'ultimo esistente fino a quello puntato, per espandere il file; se eseguite un INPUT# verrà ritornato un record nullo con un errore RECORD NOT PRESENT (record non presente). Consultate il manuale del drive per ulteriori dettagli.

ESEMPIO:

```
10 DOPEN#2,"FILE"
20 RECORD#2,10,1
30 PRINT#2, A$
40 RECORD#2,10,1
50 DCLOSE#2
```

Il programma apre un file relativo di nome FILE, si posiziona sul primo byte del record # 10, scrive il contenuto di A$, si riposiziona sul record e chiude il file.

REM

Note o commenti al programma.

REM messaggio

ESEMPIO:

10 NEXT X:REM QUESTA LINEA INCREMENTA X

RENAME

Cambia il nome di un file su disco

RENAME "vecchio nome" TO "nuovo nome" [,Ddrive]
[<ON |,>Udispositivo]

ESEMPI:

RENAME "PROVA" TO "PROVA FINALE",D0	Cambia il nome "PROVA" in "PROVA FINALE"
RENAME (A$) TO (B$),D0,U9	Cambia il nome specificato in A$ con quello in B$ sul disco 0, unità 9.

RENUMBER

Rinumera le linee di un programma BASIC

RENUMBER [nuovo numero di linea] [,incremento]
[,vecchio numero di linea]

Può rinumerare completamente od anche da una linea in poi un programma BASIC. Non è possibile usarlo per inserire in mezzo al programma delle linee in coda.

ESEMPI:

RENUMBER	Rinumera partendo da 10 a passi di 10
RENUMBER 20,20,1	Rinumera partendo da 20, a passi di 20 ed iniziando dalla linea 1, che diventa così la linea 20

RENUMBER,,65 Parte dalla vecchia linea 65, che diventa la linea 10. Se omettete un parametro dovete mettere una virgola al suo posto.

RESTORE

Ripristina il puntatore di READ in modo da poter rileggere le stesse linee DATA

RESTORE [N.o di linea]

Se RESTORE comprende un numero di linea, il puntatore viene posizionato alla linea indicata, altrimenti alla prima linea DATA del programma.

ESEMPI:

```
10 FOR I = 1 TO3
20 READ X
30 TUTTO = X + TUTTO
40 NEXT
50 RESTORE
60 GOTO 10
70 DATA 10,20,30
```

Questo programma legge i dati della linea 70, li somma ed una volta finito il FOR/NEXT ripristina il puntatore alla linea 70 e ricomincia. Premete RUN/STOP per interromperlo.

```
10 READ A,B,C
20 DATA 100,500,750
30 READ X,Y,Z
40 DATA 36,24,38
50 RESTORE40
60 READ S,P,Q
```

Il programma legge 9 valori da due linee DATA che ne comprendono 6. Ciò è possibile perchè dopo aver letto gli ultimi 3 in linea 40 ripristina (RESTORE) il puntatore alla linea iniziale, cioè la linea 20.

RESUME

Definisce la parte di programma da eseguire in caso di errore

RESUME [N.o di linea | NEXT]

Questa istruzione è usata per impedire che un errore imprevisto fermi l'esecuzione del programma. Una volta avvenuto l'errore, il controllo passa alla linea indicata dall'istruzione TRAP, l'errore viene gestito e il controllo viene ripassato ad un certo punto del programma tramite RESUME. Se RESUME non ha parametri viene rieseguita l'istruzione che ha causato l'errore. Se ha un numero di linea allora il controllo passa alla linea specificata. Se, infine, ha NEXT, viene ripresa l'esecuzione dall'istruzione successiva a quella dell'errore. Naturalmente RESUME può essere usato solo in un programma.

ESEMPIO:

```
10 TRAP 100
15 INPUT"DIGITA UN NUMERO";A
20 B = 100/A
40 PRINT"IL RISULTATO = ";B
```

```
50 INPUT"VUOI PROVARE ANCORA (S/N)";Z$:IF Z$="S"
THEN 10
60 STOP
100 INPUT"DIGITA UN ALTRO NUMERO (NON ZERO)";A
110 RESUME 20
```

Il programma chiede un numero per cui dividere 100. Se inserite 0, allora si genererà un errore e sarà chiamata la routine alla linea 100. Una volta inserito un altro numero il controllo ripasserà alla linea 20. Ciò è stato fatto tramite RESUME20, ma avreste potuto usare anche solo RESUME, che avrebbe rieseguito la divisione. Ricordate che per rientrare nel programma dalla routine di gestione dell'errore DOVETE usare RESUME e non GOTO o RETURN.

RETURN

Ritorno da subroutine

RETURN

ESEMPIO:

```
10 PRINT "PROGRAMMA PRINCIPALE"
20 GOSUB 100
30 PRINT "FINE DEL PROGRAMMA"
:
:
:
90 STOP
100 PRINT"SUBROUTINE 1"
110 RETURN
```

Il programma chiama la subroutine alla linea 100, che stampa SUBROUTINE 1 e, tramite l'istruzione RETURN, ritorna al programma principale.

RUN

Esegue un programma BASIC

1) RUN [# di linea]
2) RUN "nome del file" [,Ddrive] [<ON | ,>Udispositivo]

ESEMPI:

RUN	Inizia l'esecuzione del programma
RUN 100	Inizia l'esecuzione dalla linea 100
RUN "PRG1"	Carica da disco PRG1 e lo esegue dall'inizio
RUN (A$)	Carica ed esegue il programma in A$.

SAVE

Salva il programma su nastro o disco

SAVE ["nome del file"] [,dispositivo] [,flag EOT]

ESEMPI:

SAVE	Salva su nastro senza nome (su disco occorre sempre il nome)
SAVE"CIAO"	Salva CIAO su nastro
SAVE A$,8	Salva su disco col nome in A$ (SAVE non richiede le parentesi attorno alle variabili, al contrario di DSAVE)
SAVE "CIAO", 8	Salva CIAO su disco
SAVE "CIAO",1,2	Salva CIAO su nastro col marcatore di fine nastro (EOT).

SCALE

Stabilisce la scala dello schermo grafico

SCALE n [,Xmax,Ymax]

dove:

n = 1 (in scala) o 0 (normale)

I valori di scala possono variare da 320 a 32767 per la X e da 200 a 32767 per la Y. Con SCALE 1 senza altri parametri entrate in uno schermo virtuale da 1024x1024.

ESEMPI:

```
10 GRAPHIC 1,1
20 SCALE 1:CIRCLE 1,180,100,100,100
```

Questo programma entra in modo grafico con una scala di 1024x1024 e traccia un cerchio

```
10 GRAPHIC 3,1
20 SCALE 1,1000,5000
30 CIRCLE 1,180,100,100,100
```

Stessa funzione, ma in multicolore a 1000x5000.

Ricordate che SCALE influenza anche MOVSPR, le cui coordinate dovranno quindi essere adeguate.

SCNCLR

Pulisce lo schermo, sia testo che grafico
SCNCLR numero di modo
I modi sono i seguenti:

NUMERO MODO	MODO
0	**testo a 40 colonne**
1	**bit map**
2	**bit map e testo**
3	**bit map multicolore**
4	**bit map multicolore e testo**
5	**testo a 80 colonne**

Se omettete il numero di modo verrà cancellato lo schermo attualmente visibile: se per esempio avete bit map e testo contemporaneamente, verranno cancellati sia la bit map che lo schermo di testo. Se invece mettete il modo, verrà cancellato lo schermo corrispondente, in qualunque modo siate: così potete cancellare lo schermo grafico dal testo ad 80 colonne, o viceversa.

ESEMPI:
SCNCLR 5 Cancella le 80 colonne
SCNCLR 1 Cancella la bit map
SCNCLR 4 Cancella la bit map e il testo a 40 colonne

SCRATCH

Cancella un file dal disco

SCRATCH "nome del file" [,Ddrive] [<ON | ,>Udispositivo]

ESEMPIO:

SCRATCH "COPIA", D0
Cancella COPIA dal drive 0, unità 8.

SLEEP

Ritarda il programma per n secondi

SLEEPn
con n da 0 a 65535 secondi.

SLOW

Velocità operativa di 1MHz

SLOW

SOUND

Per ottenere effetti sonori e musicali

SOUND v,f,d [,dir] [,m] [,p] [,fdo] [,a]

dove:

v	voce (1-3)
f	frequenza (0-65535)
d	durata (0-32767)
dir	tipo di glissato (0=su; 1=giù; 2=oscillante) default 0
m	frequenza minima (0-65535) default 0
p	passo di movimento fra frequenza minima e massima e viceversa
fdo	forma d'onda (0=triangolare; 1=dente di sega; 2=pulsazione; 3=rumore) default 2
a	ampiezza della pulsazione

ESEMPI:

SOUND 1,40960,60	Suona una frequenza con la voce 1 per 1 secondo
SOUND 2,20000,50,0,2000,100	Suona dalla frequenza 2000 alla 20000 a passi di 100 e ripete per 50/60 di secondo
SOUND 3,5000,90,2,3000,500,1	Suona decrescendo dalla prima frequenza alla seconda a passi di 500 per un tempo di 90/60 di secondo.

SPRCOLOR

Predispone il multicolore 1 e 2 degli sprite

SPRCOLOR [col1] [,col2]

Questa istruzione richiede almeno uno dei due parametri, che stabiliscono rispettivamente il multicolore 1 e 2 (da 1 a 16); è possibile omettere un parametro ed inserire la virgola corrispondente se non si vuole modificarlo.

ESEMPI:

SPRCOLOR 3,7	Stabilisce il multicolore 1 a rosso e il 2 a blù
SPRCOLOR 1,2	Stabilisce l'1 a nero e il 2 a bianco.

SPRDEF

Abilita l'editor di sprite incorporato

SPRDEF

Il comando SPRDEF disegna sullo schermo una griglia larga 24 caratteri ed alta 21: ogni carattere rappresenta un punto dello sprite che viene mostrato in grandezza naturale sulla destra. Ecco un sommario delle funzioni disponibili:

COMANDO	DESCRIZIONE
1-8	**Selezionano uno sprite in risposta alla richiesta SPRITE NUMBER?**
A	**Abilita/disabilita lo spostamento automatico del cursore tracciante**
cursori	**Permettono di muoversi nell'area di disegno e non risentono di A**
«RETURN»	**A capo**
«RETURN»	**Alla richiesta SPRITE NUMBER? permette di uscire dall'editor**
HOME	**In alto a sinistra senza cancellare**
CLR	**In alto a sinistra cancellando**
1-4	**Accende/spegne i pixel**
CTRL/1-8	**Selezionano i colori da 1 a 8**
C=/1-8	**Selezionano i colori da 9 a 16**
STOP	**Ritorna a SPRITE NUMBER? senza memorizzare lo sprite**
SHIFT «RETURN»	**Ritorna a SPRITE NUMBER? memorizzando lo sprite**
X,Y	**Espandono in X e Y lo sprite**
M	**Modo multicolore**
C	**Ricopia uno sprite in un altro**

SPRITE

Stabilisce i parametri principali di uno sprite

SPRITE <numero> [,on/off] [,col] [,priorità] [,esp-x] [,esp-y] [,multi]

PARAMETRO	DESCRIZIONE
numero	**numero dello sprite (1-8)**
on/off	**abilitato/disabilitato (1/0)**
col	**colore (1-16)**
priorità	**priorità 0 lo sprite appare sopra lo sfondo, 1 appare sotto**
esp-x	**espansione in X (1/0)**
esp-y	**espansione in Y (1/0)**
multi	**multicolore (1/0)**

Se omettete qualche parametro dovete lasciare la virgola corrispondente; i parametri non specificati rimangono ai valori precedenti.

ESEMPI:

SPRITE 1,1,3	Accende lo sprite 1 col colore rosso
SPRITE 2,1,7,1,1,1	Accende lo sprite 2 in blù, lo espande in X e Y e stabilisce la priorità minore (sotto lo sfondo)
SPRITE 6,1,1,0,0,1,1	Accende lo sprite 6 in nero e multicolore, lo espande in Y e stabilisce la priorità maggiore (sopra lo sfondo). Per stabilire il multicolore 1 e 2 usate SPRCOLOR.

SPRSAV

Copia i dati di uno sprite in una stringa o viceversa, oppure da uno sprite in un altro

SPRSAV <origine>, <dest>

L'origine e la destinazione possono essere numeri di sprite o stringhe, ma non entrambe stringhe. Quando trasferite una stringa in uno sprite solo i primi 63 byte saranno usati, proprio perchè uno sprite è formato solo da 63 byte.

ESEMPI:

SPRSAV 1,A$	Trasferisce lo sprite 1 nella stringa A$
SPRSAV B$,2	Trasferisce la stringa B$ nello sprite 2
SPRSAV 2,3	Trasferisce lo sprite 2 nello sprite 3.

SSHAPE/GSHAPE

Salva/carica forme in/da variabili stringa.

SSHAPE e GSHAPE sono usate per caricare e salvare aree rettangolari dello schermo grafico da/in variabili stringa. Questo è il comando per salvare un'area:

SSHAPE stringa, X1, Y1 [,X2,Y2]

dove:

stringa	La variabile dove salvare l'area
X1,Y1	Le coordinate di un angolo dell'area (in scala)
X2,Y2	Le coordinate dell'angolo opposto (se omesse vale il cursore pixel)

Questo invece è il comando per caricare un'area da una stringa:

GSHAPE stringa [,X,Y][,modo]

dove:

stringa	La variabile da cui caricare l'area
X,Y	Le coordinate dell'angolo superiore sinistro (in scala; se omesse vale il CP)
modo	Tipo di tracciamento: 0=uguale all'originale (default) 1=punti invertiti (negativo) 2=OR logico tra la forma e l'area 3=AND logico 4=XOR logico

I modi disponibili vi permettono di tracciare la forma uguale all'originale, invertita o mischiata all'area di disegno.

ESEMPI:

SSHAPE A$,10,10	Salva in A$ l'area rettangolare dalle coordinate 10,10 al CP
SSHAPE B$,20,30,43,50	Salva l'area da 20,30 a 43,50
SSHAPE D$,+10,+10	Salva l'area dal CP a 10 pixel a destra e in basso

GSHAPE A$,120,20	Preleva da A$ una forma piazzandola da 120,20
GSHAPE B$,30,30,1	Piazza una forma da 30,30 invertendola
GSHAPE C$,+20,+30	Piazza una forma partendo da 20 pixel a destra e 30 in basso del CP

NOTA: Attenzione con i modi da 1 a 4 e la grafica multicolore: i risultati possono essere imprevedibili.

STASH

Trasferisce dalla memoria centrale all'espansione di memoria

STASH # byte,indint,indest,bn
Riferitevi al comando FETCH per i parametri.

STOP

Ferma l'esecuzione del programma
STOP

SWAP

Scambia i contenuti della memoria centrale con quella dell'espansione

SWAP # byte,indint,indest,bn
Riferitevi al comando FETCH per i parametri.

SYS

Esegue un programma in linguaggio macchina partendo da un indirizzo specificato

SYS indirizzo [,a] [,x] [,y] [,s]

Quest'istruzione chiama una routine ad un indirizzo specificato nella configurazione di memoria selezionata precedentemente tramite il comando BANK. Opzionalmente si possono caricare i registri A,X, Y e S (o P, registro di stato) del processore prima di chiamare la routine.

Il campo dell'indirizzo varia da 0 a 65535 e si usa BANK per indirizzare tutta la memoria disponibile.

ESEMPI:

SYS32768	Chiama la routine che parte da 32768
SYS6144,0	Chiama la routine in 6144 caricando 0 nel registro A (Accumulatore).

TEMPO

Definisce la velocità delle note di un'istruzione PLAY

TEMPOn

con n (durata relativa) da 1 a 255.
Il default è 8 e la durata delle note aumenta proporzionalmente con n.

ESEMPI:

TEMPO 16	Definisce la durata a 16
TEMPO 1	Durata minima
TEMPO 250	Durata molto lunga.

TRAP

Rileva e gestisce gli errori del programma BASIC che sta girando al momento

TRAP [# di linea]

Se omettete il # di linea allora verrà disattivata la gestione degli errori, altrimenti il controllo del programma passerà al numero di linea specificato. Per uscire dalla routine di gestione degli errori usate RESUME. Attenzione: un errore nella routine di gestione dell'errore sarà fatale. Guardate anche le variabili di sistema ST, EL, ER, DS e DS$.

ESEMPIO:

.100 TRAP 1000	A 1000 si gestisce l'errore
:	Stampa il messaggio d'errore col numero di linea
:	
.1000?ERR$(ER);EL	
1010 RESUME	Riprende l'esecuzione del programma.

TROFF

Disattiva il modo "trace"

TROFF

TRON

Abilita il modo ''trace''

TRON

TRON si usa nella messa a punto di un programma e visualizza tra parentesi quadre il numero della linea di programma in esecuzione, permettendo così di osservare il corso delle decisioni.

VERIFY

Confronta il programma in memoria con quello su nastro o disco

VERIFY "nome del file" [,dispositivo] [,flag di rilocazione]

Usatelo dopo il SAVE di un programma.

ESEMPI:

VERIFY	Confronta il prossimo programma su nastro
VERIFY ''CIAO''	Confronta su nastro il programma CIAO
VERIFY ''CIAO'',8,1	Confronta su disco CIAO dalla zona originaria di SAVE.

VOL

Definisce il volume del suono per le istruzioni SOUND e PLAY

VOL livello (0-15)

ESEMPI:

VOL 0	Azzera il volume
VOL 15	Pone il volume al massimo

WAIT

Ferma l'esecuzione del programma fino al verificarsi di una certa condizione

WAIT <locazione>, <maschera 1> [,maschera 2]

dove:

locazione	= 0-65535
maschera	= 0-255

L'istruzione WAIT sospende l'esecuzione del programma finchè un certo indirizzo di memoria non corrisponde alla maschera di bit fornita nell'istruzione. La maschera #2 è opzionale: se presente essa esegue un XOR del contenuto della locazione prima di effettuare un AND con la maschera#1, altrimenti viene effettuato solo l'AND.

ESEMPI:

WAIT 1,16,16	Attende la pressione di un tasto del registratore
WAIT 53273,2	Attende una collisione sprite/fondo

WIDTH

Dispone la larghezza delle linee tracciate

WIDTHn

ESEMPI:

WIDTH 1	Larghezza 1 (normale)
WIDTH 2	Larghezza 2 (doppia)

WINDOW

Definisce una finestra di testo

WINDOW C,R,C1,R1 [,cancellazione]

Il comando WINDOW definisce una finestra logica di testo sullo schermo a 40 od 80 colonne; C e C1 variano da 0-39 per le 40 colonne e da 0-79 per le 80; R e R1 da 0-24 per entrambi gli schermi. L'opzione di cancellazione (0/1) pulisce la finestra se 1.

ESEMPI:

WINDOW 5,5,35,20	Definisce una finestra da 5,5 a 35,20
WINDOW 10,2,33,24,1	Definisce e pulisce una finestra da 10,2 a 33,24.

FUNZIONI DEL BASIC

Il formato della descrizione è:

FUNZIONE (argomento)

dove l'argomento può essere un valore numerico, variabile o stringa. Ogni funzione descritta è seguita da un esempio: la prima linea in grassetto è la funzione richiesta; la seconda linea è la risposta del computer.

ABS

Ritorna il valore assoluto dell'argomento

ABS (X)

ESEMPIO:

```
PRINT ABS (7*(-5))
35
```

ASC

Ritorna il codice CBM ASCII del primo carattere di X$

ASC (X$)

ESEMPIO:

```
X$ = "C128":PRINT ASC(X$)
67
```

ATN

Ritorna l'arcotangente di X in gradi radianti (da -pi/2 a pi/2)

ATN (X)

ESEMPIO:

```
PRINT ATN (3)
1.24904577
```

BUMP

Ritorna informazioni sulle collisioni degli sprite

BUMP (N)

Per determinare quali sprite abbiano colliso dall'ultimo controllo, usate BUMP. BUMP(1) registra le collisioni sprite/sprite, mentre BUMP(2) quelle sprite/fondo. Non è necessario abilitare COLLISION per usare BUMP. Le posizioni dei bit (0-7) nel valore ritornato da BUMP corrispondono rispettivamente agli sprite da 1-8; dopo ogni lettura BUMP si azzera fino ad una nuova collisione (infatti questa funzione non fa altro che leggere i registri di collisione sprite/sprite e sprite/fondo del VIC II, rispettivamente in 53278 ($D01E) e 53279 ($D01F) che si azzerano dopo ogni lettura e riportano nei bit da 0-7 lo stato degli sprite 0-7).

Questa tabella riporta il valore corrispondente per ogni sprite:

Valore di BUMP	128	64	32	16	8	4	2	1
Sprite	8	7	6	5	4	3	2	1

Il valore relativo ad ogni sprite può essere ricavato anche con: 2 ↑ (N-1). N è il numero di sprite (1-8)

ESEMPI:

PRINTBUMP (1) 12 indica che gli sprite 3 e 4 hanno colliso
PRINTBUMP (2) 32 indica che lo sprite 6 ha colliso con lo sfondo (si possono anche avere dei valori che indicano che più sprite hanno colliso con lo sfondo)

CHR$

Ritorna il carattere di codice X

CHR$ (X)

L'argomento X va da 0-255; questa funzione è l'opposto di ASC; per vedere i codici di ogni carattere consultate l'Appendice E.

ESEMPI:

PRINT CHR$ (65) Stampa il carattere A
PRINT CHR$ (147) Pulisce lo schermo di testo (come CLR/HOME)

COS

Ritorna il coseno dell'angolo X in radianti

COS (X)

ESEMPIO:

```
PRINT COS (π/3)
.5
```

FNxx

Ritorna il valore della funzione definita dall'utente

FNxx (X)

Questa funzione ritorna il valore della funzione precedentemente creata con DEFFNxx per il parametro X

ESEMPIO:

```
10 DEF FNAA (X) = (X-32)*5/9
20 INPUT X
30 PRINT FNAA(X)
RUN
?40 (? è la richiesta di input)
4.44444445
```

NOTA: Se usate il comando GRAPHIC nel programma definite la vostra funzione dopo questo comando, altrimenti la definizione della funzione sarà distrutta.

FRE

Ritorna l'ammontare in byte della memoria libera

FRE (X)

L'argomento X determina il banco di memoria RAM da controllare; X = 0 per la memoria BASIC e X = 1 per la memoria delle variabili (normalmente FRE(0)=58109 e FRE(1)=64256)

HEX$

Converte un numero decimale in una stringa esadecimale

HEX$ (X)

ESEMPIO:

```
PRINT HEX$(53280)
D020
```

INSTR

Ritorna la posizione di una stringa all'interno di un'altra

INSTR (stringa 1,stringa 2[,posizione di partenza])

Il numero intero ritornato indica la posizione da cui parte la stringa 2 nella stringa 1; il terzo parametro indica la posizione da cui iniziare la ricerca nella stringa 1; se INSTR=0 allora la stringa 2 non esiste nella stringa 1.

ESEMPIO:

```
PRINT INSTR("COMMODORE 128","128") 11
```

INT

Ritorna la parte intera di un valore reale

INT (X)

Questa funzione ritorna la parte intera di un valore positivo o negativo. Se positivo si avrà il troncamento dei decimali; se negativo l'arrotondamento all'intero negativo più piccolo.

ESEMPI:

```
PRINT INT(3.14)
3
PRINT INT(-3.14)
-4
```

JOY

Ritorna la posizione del joystick e lo stato del pulsante

JOY (N)

dove N vale:

1 per il joystick 1
2 per il joystick 2

Ogni valore di JOY maggiore di 127 indica la pressione del pulsante di fuoco. Le direzioni sono indicate nel modo seguente:

```
          1
      8       2
  7       0       3
      6       4
          5
```

ESEMPI:

PRINT JOY(2) 135	Il joystick 2 spara a sinistra
IF JOY (1)>127 THEN PRINT"FUOCO"	Determina la pressione del pulsante
DIR = JOY (1) AND 15	Ritorna solo la direzione del joystick 1.

LEFT$

Ritorna i caratteri più a sinistra della stringa

LEFT$ (stringa,intero)

ESEMPIO:

```
PRINT LEFT$ ("COMMODORE",5)
COMMO
```

LEN

Ritorna la lunghezza di una stringa

LEN (stringa)

Il valore ritornato va da 0-255.

ESEMPIO:

PRINT LEN ("COMMODORE128")
12

LOG

Ritorna il logaritmo naturale (o neperiano) di X con X>0.

LOG (X)

ESEMPIO: PRINT LOG (37/5)
2.00148

Se volete ottenere il logaritmo decimale (o volgare) di X, usate LOG(X)/LOG(10)

MID$

Ritorna/inserisce una sottostringa da/in una stringa maggiore

MID$ (stringa,posizione di partenza [,lunghezza])

Usate questa funzione per estrarre od inserire una stringa minore da/in una stringa maggiore.

ESEMPI:

PRINT MID$ ("COMMODORE 128",3,5)
MMODO
A$="COMMODORE 128":MID$ (A$,11,3) = "64 ":PRINT A$ COMMODORE 64

Non potete naturalmente inserire stringhe che oltrepassino la stringa iniziale.

PEEK

Ritorna il contenuto di una specifica locazione di memoria

PEEK (X)

Il dato sarà prelevato dalla configurazione di memoria selezionata dall'ultimo comando BANK.

ESEMPIO:

```
10 BANK 15:VIC = DEC("D000")
20 FOR I = 1 TO 47
30 PRINT PEEK(VIC + I)
40 NEXT
```

Questo esempio mostra il contenuto dei registri del chip VIC II.

PEN

Ritorna le coordinate X,Y della penna ottica

PEN (N)

dove: **N** = 0,1 per X,Y a 40 colonne
N = 2,3 per X,Y a 80 colonne
N = 4 per il segnale di lettura valida (solo a 80 colonne)

I valori a 40 colonne non sono scalati e usano lo stesso piano di coordinate degli sprite (24,50-343,249); purtroppo la X deve essere contenuta in 8 bit e quindi si ha una risoluzione di 160 punti contro 320: attenzione che PEN(0) moltiplica per 2 il valore letto per riportarlo alla scala normale, e così la X varierà da 48-367 invece che da 24-343; PEN(1)non dà invece problemi. I valori a 80 colonne variano invece da 0-79 e da 0-24 ed indicano quindi colonna e riga di un carattere piuttosto che un pixel. Questi valori sono validi solo quando PEN(4)=1. I valori letti sia a 40 che a 80 colonne sono alquanto instabili e variano a seconda della luminosità dello schermo e della qualità della penna.

La penna ottica va installata nella porta 1.

ESEMPI:

10 PRINT PEN(0) PEN(1)	Mostra le coordinate X,Y a 40 colonne
10 DO UNTIL PEN(4):LOOP	Attende la validità dei valori a 80 colonne
20 X = PEN(2)	
30 Y = PEN(3)	
40 REM:RESTO DEL PROGRAMMA	

π

Ritorna il valore di pi (3.14159265)

π

ESEMPIO:

PRINT π Risulta 3.14159265

POINTER

Ritorna l'indirizzo di una variabile

POINTER (variabile)

Questa funzione vale 0 se la variabile non è stata ancora definita.

ESEMPIO:

A = POINTER (Z)	Questo esempio ritorna l'indirizzo nella RAM del banco 1 della variabile Z.

POS

Ritorna la colonna corrente del cursore di testo.

POS (X)

La funzione POS, per X fittizio da 0-255, ritorna la colonna del cursore, da 0-39 per 40 colonne e da 0-79 per le 80.

ESEMPIO:

```
FOR I= 1 TO 10:?SPC(I)POS(0):NEXT
```

POT

Ritorna il valore delle paddle

POT (N)

con

N = 1-4 per le paddle 1-4 (1,2 in porta 1, 3,4 in porta 2).

I valori sono nel campo 0-255; valori maggiori indicano la pressione del pulsante.

ESEMPIO:

```
10 PRINT POT(1)
20 IF POT (1) > 255 THENPRINT"FUOCO"
```

Ciò mostrerà il valore della paddle 1, compreso il pulsante.

RCLR

Ritorna il colore di un N.o di colore

RCLR(N)

dove N=0-6 corrisponde al N.o di registro colore dell'istruzione COLOR. Riferitevi a questa o alla tabella seguente per le specifiche di N.o.

COL. SORGENTE	DESCRIZIONE
0	Fondo a 40 colonne
1	Primo piano a 40 colonne
2	Multicolore 1
3	Multicolore 2
4	Bordo a 40 colonne
5	Carattere a 40 e 80 colonne
6	Fondo a 80 colonne

ESEMPIO:

```
10 FOR I = 0 TO 6
20 PRINT"N.O DI COLORE"I"CORRISPONDE AL COLORE"RCLR(I)
30 NEXT
```

Questo programma stampa i colori corrispondenti ai N.i di colore.

RDOT

Ritorna la posizione ed il colore corrente del cursore pixel

RDOT (N)

dove N=0,1 per le coordinate X,Y del CP e N=2 per il # di colore (0-3)

ESEMPI:

PRINT RDOT (0)	Legge la X
PRINT RDOT (1)	Legge la Y
PRINT RDOT (2)	Legge il # di colore.

RGR

Ritorna il modo grafico corrente

RGR (X)

L'argomento X è fittizio da 0-255 e deve comunque essere specificato. Il parametro intero ritornato varia da 0-6 e corrisponde ai parametri di GRAPHIC. Riferitevi a quest'ultima per le specifiche.

VALORE	MODO GRAFICO
0	**40 colonne (VIC) modo testo**
1	**Bit map standard**
2	**Bit map schermo grafico + testo**
3	**Bit map multicolor**
4	**Bit map multicolor + testo**
5	**80 colonne (8563) modo testo**

ESEMPI:

PRINT RGR (0)	Mostra il modo grafico corrente, in questo caso bit map standard
1	
PRINT RGR (0)	
8	

8 corrisponde a bit map multicolore (3) + testo a 80 colonne (5)

RIGHT$

Ritorna i caratteri più a destra di una stringa

RIGHT$ (stringa,intero)

ESEMPIO:

```
PRINT RIGHT$("BASEBALL",5)
EBALL
```

RND

Ritorna un numero casuale

RND (X)

dove: X=0 per usare come seme il timer interno al CIA N.1 X>0 per rielaborare il seme precedente X<0 per usare come seme il valore assoluto di X (per gli stessi valori negativi di X si ottengono quindi gli stessi numeri casuali)

ESEMPI:

PRINT RND(0)	Ritorna un numero fra 0 e 1 escluso
.507824123	
PRINT INT(RND(1)*100+1)	Mostra un numero fra 1 e 100 compreso (0 escluso)
89	

RSPCOLOR

Ritorna i colori multicolore 1 e 2 degli sprite

RSPCOLOR(X)

dove X=1,2 per leggere il multicolore 1 o 2. Il valore ritornato è un intero fra 1 e 16.
Tale funzione è il contrario di SPRCOLOR. Vedere anche questo comando.

ESEMPIO:

```
10  SPRITE 1,1,2,0,1,1,1
20  SPRCOLOR 5,7
30  PRINT"IL MULTICOLORE SPRITE N.1 E'"RSPCOLOR(1)
40  PRINT"IL MULTICOLORE SPRITE N.2 E'"RSPCOLOR(2)
RUN
IL MULTICOLORE SPRITE N.1 E' 5
IL MULTICOLORE SPRITE N.2 E' 7
```

Il programma accende in bianco lo sprite 1, assegna i colori porpora e blù e poi rilegge e stampa i colori multicolore tramite RSPCOLOR.

RSPPOS

Ritorna la velocità e la posizione di uno sprite

RSPPOS (N.o sprite,posizione | velocità)

dove N.o sprite (1-8) specifica lo sprite e il secondo parametro (0-2) legge la X (0), la Y (1) o la velocità (2).

ESEMPIO:

```
10 SPRITE 1,1,2
20 MOVSPR 1,45#13
30 PRINT RSPPOS (1,0) RSPPOS(1,1) RSPPOS(1,2)
```

Questo programma stampa la posizione X,Y e la velocità dello sprite N.1.

RSPRITE

Ritorna le caratteristiche dello sprite

RSPRITE(N.o sprite,caratteristica)

Specificate il N.o di sprite e come caratteristica un intero da 0-5. La funzione ritornerà:

CARATTERISTICA	VALORE
0	Acceso/spento (1/0)
1	Colore (1-16)
2	Priorità (0/1)
3	Espansione X (0/1)
4	Espansione Y (0/1)
5	Multicolore (0/1)

ESEMPIO:

```
10 FOR I = 0 TO5
20 PRINTRSPRITE(1,I)
30 NEXT
```

Questo programma stampa le 6 caratteristiche dello sprite N.1 .

RWINDOW

Ritorna le dimensioni della finestra corrente o le colonne (40/80) dello schermo selezionato.

RWINDOW(N)

dove N vale: 0 per il numero di righe; 1 per il numero di colonne; 2 per lo schermo (40/80)
Il contrario della funzione RWINDOW è il comando WINDOW.

ESEMPIO:

```
10 WINDOW 1,1,10,10
20 PRINT RWINDOW(0)RWINDOW(1)RWINDOW(2)
RUN
9 9 40
```

Il programma riporta le righe (10, 0-9), le colonne (9, 0-9) e la taglia dello schermo (40 colonne).

SGN

Ritorna il segno dell'argomento

SGN (X)

ESEMPIO:

```
PRINT SGN (4.5) SGN(0) SGN(-2.3)
1 0 -1
```

SIN

Ritorna il seno dell'argomento

SIN (X)

ESEMPIO:

```
PRINTSIN (π/3)
.866025404
```

SPC

Spazia di X nella stampa

SPC (X)

ESEMPIO:

```
PRINT"COMMODORE"SPC(3)"128"
COMMODORE   128
```

SQR

Ritorna la radice quadrata dell'argomento

SQR (X)

ESEMPIO:

```
PRINT SQR(25)
5
```

STR$

Trasforma un tipo numerico in tipo stringa

STR$(X)

ESEMPI:

```
PRINT STR$ (123.45)
123.45
PRINT STR$(-89.03)
-89.03
PRINT STR$(1E20)
1E+20
```

TAB

Esegue la tabulazione del cursore

TAB (X)

ESEMPIO:

```
10 PRINT "COMMODORE"TAB(25)"128"
COMMODORE  128
```

Questa funzione è diversa da SPC, poichè non fa avanzare il cursore di X spazi, ma lo posiziona sulla colonna X.

TAN

Ritorna la tangente dell'argomento, con X in gradi radianti

TAN (X)

ESEMPIO:

```
PRINT TAN(.785398163)
1
```

USR

Chiama un sottoprogramma in linguaggio macchina definito dall'utente

USR (X)

Quando usate questa funzione il programma BASIC chiama un programma in linguaggio macchina il cui indirizzo di partenza si trova nelle locazioni 4633-4634 ($1219-$121A) per il Modo C128, e nelle locazioni 785-786 ($0311-$0312) per il Modo C64. Il parametro X viene passato al linguaggio macchina tramite l'accumulatore reale ($63-$68 per il C128), e sempre tramite questo accumulatore verrà ritornato un altro parametro che andrà nella variabile corrispondente alla funzione. Questo è un comodo metodo di scambiare variabili fra BASIC e linguaggio macchina. Prima di chiamare la funzione dovete inserire in 4633-4634 (o 785-786 per il C64) l'indirizzo del programma.

ESEMPIO:

```
 10 POKE 4633,0
20 POKE 4634,48
30 A = USR(X)
40 PRINT A
```

Il programma chiama una funzione in 12288 e memorizza il risultato in A. Il vettore della funzione deve essere nel banco 15,e la funzione stessa deve risiedere nel banco 0 al di sotto di 16384.

VAL

Converte una stringa in un valore numerico. E' la funzione opposta di STR$

VAL (X$)

ESEMPIO:

```
10 A$ = "120"
20 B$ = "365"
30 PRINT VAL (A$+B$)
RUN
485
```

XOR

Esegue l'OR esclusivo di due valori

XOR (N1,N2)

Questa funzione ritorna l'OR esclusivo di N1 e N2. X = XOR (N1,N2) dove N1 e N2 sono valori assoluti da 0-65535

ESEMPIO:

```
PRINT XOR (128,64)
192
```

PAROLE DI SISTEMA RISERVATE (PAROLE CHIAVE)

Questa sezione elenca le parole costituenti il BASIC 7.0. Queste parole possono essere usate solo come componenti del BASIC, eccetto che tra virgolette, per esempio in un'istruzione PRINT.

ABS
AND
APPEND
ASC
ATN
AUTO
BACKUP
BANK
BEGIN
BEND
BLOAD
BOOT
BOX
BSAVE
BUMP
CATALOG
CHAR
CHR$
CIRCLE
CLOSE
CLR
CMD
COLLECT
COLLISION
COLOR
CONCAT
CONT
COPY
COS
DATA
DCLEAR
DCLOSE
DEC
DEFFN
DELETE
DIM
DIRECTORY
DLOADDO
DOPEN
DRAW
DS
DS$
DSAVE
DVERIFY
EL
ELSE
END
ENVELOPE
ER
ERR$
EXIT
EXP
FAST
FETCH
FILTER
FN
FOR
FRE
GET
GET#
GO64
GOSUB
GOTO
GO TO
GRAPHIC
GSHAPE
HEADER
HELP
HEX$
IF
INPUT
INPUT#
INSTR
INT
JOY
KEY
LEFT$
LEN
LET
LIST
LOAD
LOCATE
LOG
LOOP
MID
MONITOR
MOVSPR
NEW
NEXT
NOT
ON
OPEN
OR
PAINT
PEEK
PEN
PLAY
POINTER
POKE
POS
POT
PRINT
PRINT#
PUDEF
RCLR
RDOT
READ
RECORD
REM
RENAME

RENUMBER	**SLEEP**	**TEMPO**
RESTORE	**SLOW**	**THEN**
RESUME	**SOUND**	**TI**
RETURN	**SPC**	**TI$**
RGR	**SPRCOLOR**	**TO**
RIGHT$	**SPRDEF**	**TRAP**
RND	**SPRITE**	**TROFF**
RREG	**SPRSAV**	**TRON**
RSPCOLOR	**SQR**	**UNTIL**
RSPPOS	**SSHAPE**	**USING**
RSPRITE	**ST**	**USR**
RUN	**STASH**	**VAL**
RWINDOW	**STEP**	**VERIFY**
SAVE	**STOP**	**VOL**
SCALE	**STR$**	**WAIT**
SCNCLR	**SWAP**	**WHILE**
SCRATCH	**SYS**	**WIDTH**
SGN	**TAB**	**WINDOW**
SIN	**TAN**	**XOR**

> **NOTA:** Le parole QUIT e OFF esistono ma non sono state implementate nel BASIC 7.0

I nomi di variabili riservate sono DS, DS$, ER, EL, ST, TI, TI$ e la funzione ERR$. Parole chiave come TO e IF od ogni altro nome che contenga parole chiave, tipo RUN, NEW o LOAD non possono essere usate.

ST è una variabile di stato relativa all'I/O (eccetto tastiera e schermo). Il valore di ST dipende dal risultato dell'ultima operazione di I/O effettuata. In generale, se ST vale 0 l'operazione ha avuto successo.

TI e TI$ sono variabili collegate al clock in tempo reale incorporato nel Commodore 128. Questo clock è aggiornato ogni 1/60 di secondo. Parte da zero all'accensione, e si può modificare cambiando il valore di TI$. La variabile TI dà il tempo in 1/60 di secondo e non può essere cambiata. TI$ invece è una stringa che dà il tempo in formato a 24 ore, e può essere cambiata: i primi due caratteri sono le ore (00-23), poi vengono i minuti (00-59) ed infine i secondi (00-59). Una volta immesso un orario, questo verrà assegnato automaticamente al contatore dell'orologio.

ESEMPIO:

TI$="101530" Pone l'orologio alle 10:15 e 30 secondi Anti Meridiane (AM)

Il valore dell'orologio viene perso allo spegnimento della macchina. Esso riparte da zero all'accensione, quando arriva a 235959 oppure si assegna TI$="000000"

La variabile DS legge il canale di comando del drive e ne ritorna lo stato corrente. Per avere quest'informazione come messaggio usate PRINTDS$. Queste variabili di stato si usano quando la luce del drive lampeggia per scoprire il tipo di errore avvenuto.

ER, EL e la funzione ERR$ sono usate nella gestione degli errori BASIC. Solitamente sono utili solo da programma. ER contiene il numero dell'errore, EL il numero di linea che lo ha originato ed ERR? il messaggio d'errore. Con PRINT ERR$ (ER) stampate il messaggio d'errore.

SIMBOLI DI SISTEMA RISERVATI

I seguenti caratteri sono simboli di sistema riservati.

	SIMBOLO	SIGNIFICATO (I)
+	Segno più	Addizione; concatenazione di stringhe; movimento grafico relativo; numero decimale nel monitor di linguaggio macchina
-	Segno meno	Sottrazione; numeri negativi; negazione; movimento relativo di sprite
*	Asterisco	Moltiplicazione aritmetica
/	Slash	Divisione aritmetica
↑	Freccia su	Elevamento a potenza
	Spazio	Separatore fra parole chiave e variabili (opzionale)
=	Uguale	Assegnazione di valori; confronto relazionale
<	Minore di	Confronto relazionale
>	Maggiore di	Confronto relazionale
,	Virgola	Formato di stampa; separatore dei parametri
.	Punto	Punto decimale nelle costanti reali
;	Punto e virgola	Formato di stampa; separatore dei parametri
:	Due punti	Separatore delle istruzioni BASIC in una linea di programma
“”	Virgolette	Delimitatori delle costanti stringa
?	Punto domanda	Abbreviazione di PRINT
()	Parentesi	Valutazione di espressioni e funzioni
%	Percento	Dichiarazione di tipo intero; numero binario nel monitor
#	Numero	Precede il numero di file logico nelle istruzioni di I/O
$	Dollaro	Dichiarazione di tipo stringa; numero esadecimale nel monitor
&	E commerciale	Numero ottale nel monitor
π	Pigreco	Costante numerica 3.141592654

4

UN PASSO OLTRE IL NORMALE BASIC

Questo capitolo ci porta un passo oltre il semplice BASIC e presenta una raccolta di utili routine. Potete incorporare queste routine nei vostri programmi secondo la necessità. In molti casi basterà cambiare soltanto i numeri di linea.

CREARE UN MENÙ

Un menù è una lista di scelte che voi selezionate per eseguire un'operazione specifica in un programma applicativo. Un menù indirizza il computer verso una certa parte di un programma. Ecco un esempio generale di menù:

```
5 REM COMPOSIZIONE DI UN MENÙ
10 SCNCLR0
20 PRINT"1. PRIMA SCELTA"
30 PRINT"2. SECONDA SCELTA"
40 PRINT"3. TERZA SCELTA"
50 PRINT"4. QUARTA SCELTA"
100 PRINT:PRINT"SELEZIONA UN'OPERAZIONE DA QUELLE SOPRA"
110 GETKEYA$
120 A=VALA$:IFA>4THEN10
130 ONAGOSUB1000,2000,3000,4000
140 GOTO10:REM RITORNO AL MENÙ
999 STOP
1000 REM QUI INIZIA LA ROUTINE RELATIVA ALLA PRIMA SCELTA
1999 RETURN
2000 REM QUI INIZIA LA SECONDA ROUTINE
2999 RETURN
3000 REM QUI INIZIA LA TERZA ROUTINE
3999 RETURN
4000 REM QUI INIZIA LA QUARTA ROUTINE
4999 RETURN
```

Programma 4-1. Composizione di un menù

Il comando SCNCLR0 della linea 10 pulisce lo schermo a 40 colonne (usate SCNCLR5 se siete nello schermo a 80 colonne). La selezione più semplice è tramite un numero. Potete usare tante selezioni quante ne stanno sullo schermo. La linea 100 mostra un messaggio all'utente. Il comando GETKEY nella linea 110 forza il computer ad aspettare la pressione di un tasto. A è una variabile stringa, poichè un tasto rappresenta un carattere. Nella linea 120 la stringa è convertita in un numero dalla funzione VAL, in modo da poter essere interpretata come un valore numerico dall'istruzione ON GOTO. Nella linea 120 l'istruzione IF...THEN previene gli errori dell'utente impedendogli di selezionare un numero fuori dal campo delle scelte (4). La linea 130 dirige il controllo alla sezione appropriata (cioè al numero di linea) del vostro programma. Poichè questo esempio offre quattro scelte, dovete includere almeno quattro numeri di linea. La linea 1999 ritorna al menù alla fine di ogni subroutine che aggiungerete alle linee 1000, 2000, 3000, 4000 nella composizione del menù.

ROUTINE DEL BUFFER DI TASTIERA

Il buffer (serbatoio) di tastiera del C128 può contenere e fornire fino a 10 caratteri presi dalla tastiera. Questo è utile in un programma di scrittura dove può succedere, in certi momenti, di scrivere più velocemente di quanto il programma possa ricevere. I caratteri che non sono stati ancora stampati sono memorizzati temporaneamente nel buffer di tastiera. Il computer può mantenere nel buffer la prossima istruzione per usarla quando il programma sarà pronto. Questo buffer permette un massimo di 10 caratteri in fila. Per vedere il buffer in funzione immettete il comando SLEEP5 e subito dopo premete una decina di tasti. Dopo 5 secondi tutti i dieci tasti saranno mostrati sullo schermo. Ecco una routine di uso del buffer che vi permette, da programma, di porre dei caratteri nel buffer in modo da fornirli automaticamente al computer non appena finita l'esecuzione del programma. Nella linea 10 nella locazione di memoria 208 (198 in Modo C64) viene posto un numero tra 0 e 10 - il numero di caratteri presenti nel buffer di tastiera. Nella linea 20 le locazioni di memoria da 842 a 851 (631-640 in Modo C64) vengono riempite con qualunque carattere si voglia. Nell'esempio si hanno sette caratteri di ritorno a capo nel buffer. CHR(13) è il codice del carattere di ritorno a capo. La linea 40 stampa "?CHR(156)" ma non esegue il comando. La linea 50 stampa "LIST". Nessuno dei due comandi è eseguito finchè il programma non finisce. Sul C128, il buffer di tastiera si svuota automaticamente alla fine del programma. Così, i caratteri nel buffer (ritorno a capo) sono prelevati ed agiscono come se voi steste premendo il tasto «RETURN». Quando il ritorno a capo viene eseguito su una linea di schermo che mostra i comandi stampati dalle linee 40 e 50 anche i comandi vengono eseguiti come se li aveste scritti voi direttamente. Alla fine del programma il colore dei caratteri diventa porpora e il programma viene LISTato sullo schermo. Questa tecnica è comoda per rieseguire un programma (con RUN o GOTO). La prossima sezione dà un esempio pratico dell'uso della routine.

```
10 POKE208,7: REM SPECIFICA IL # DEI CARATTERI NEL BUFFER
20 FORI=842TO849:POKEI,13:NEXT:REM PONE I CARATTERI NEL BUFFER
30 SLEEP2: REM RITARDO
40 SCNCLR:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT"?CHR$(156)"
50 PRINT:PRINT:PRINT:PRINT"LIST": REM STAMPA LIST SULLO SCHERMO
60 PRINTCHR$(19):PRINT:PRINT:REM VA IN CIMA ALLO SCHERMO E GIÙ DUE LINEE
70 REM FINITO IL PROGRAMMA, IL BUFFER SI SVUOTA E VENGONO ESEGUITI 7
RETURN
80 REM CIÒ CAMBIA IL COLORE IN PORPORA E LISTA IL PROGRAMMA AUTOMATICA-
MENTE
90 REM COME SE AVESTE PREMUTO IL TASTO <RETURN>
```

Programma 4-2. Routine di uso del buffer

ROUTINE DI CARICAMENTO

Il buffer può essere usato dalle routine di caricamento automatico. Molti programmi spesso implicano il caricamento di parecchie routine in linguaggio macchina come anche di programmi BASIC. I risultati del caricatore seguente sono simili a quelli trovati in molti programmi commerciali.

```
2 COLOR4,1:COLOR0,1:COLOR5,1
5 A$="DISEGNO"
10 SCNCLR:PRINT:PRINT:PRINT:PRINT"LOAD"CHR$(34)A$CHR$(34)",8,1"
15 PRINT:PRINT:PRINT"NEW"
25 B$="FILE3.BIN"
30 PRINT:PRINT:PRINT"LOAD"CHR$(34)B$CHR$(34)",8,1"
45 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT"SYS12*256"
90 PRINTCHR$(5):PRINT"          SALUTI DA COMMODORE"
100 PRINT"     ASPETTATE PER FAVORE - CARICAMENTO IN CORSO"
:PRINTCHR$(144)
200 PRINTCHR$(19)
300 POKE208,7:FORI=842TO851:POKEI,13:NEXT
```

Programma 4-3. Routine di caricamento

La linea 2 colora il bordo, il fondo ed i caratteri in nero. La linea 5 assegna ad A$ il nome "DISEGNO", che in questo esempio si assume essere un file binario di 8 Kbyte rappresentante uno schermo grafico. La linea 10 stampa l'istruzione LOAD per il disegno senza eseguirla. A ciò penserà il ritorno a capo nel buffer di tastiera, una volta finito il programma. La linea 15 stampa NEW, che sarà esguito tramite il ritorno a capo nel buffer. Dopo aver caricato un programma in linguaggio macchina, NEW rimette i puntatori ai valori originari pulendo l'area delle variabili. La linea 30 mostra la seconda istruzione per il caricamento del file in linguaggio macchina "FILE3.BIN". Quest'ultimo è un ipotetico programma che abilita la pagina grafica chiamata DISEGNO e fa qualunque altra cosa vogliate. La linea 45 inizia (SYS12*256) l'esecuzione del programma "FILE3.BIN" a 3072 (0C00) non appena si svuota il buffer di tastiera. Questo è solo un esempio da seguire fatto apposta per voi. "DISEGNO" e "FILE3.BIN" sono programmi forniti da voi e servono solo ad illustrare una tecnica di caricamento automatico. Poichè il precedente colore dei caratteri era nero, tutte le istruzioni di caricamento sono state stampate in nero su sfondo nero, e perciò invisibili. Il CHR(5) della linea 90 cambia il colore dei caratteri in bianco, così le sole scritte visibili sono quelle in bianco delle linee 90 e 100, mentre il drive sta caricando "DISEGNO" e "FILE3.BIN". La linea 300 è la routine del buffer. Se aveste dovuto fare ogni passo manualmente avreste dovuto premere 7 volte il tasto «RETURN». Questo programma pone 7 caratteri di ritorno a capo nel buffer per farli poi prelevare alla fine. Per ogni RETURN prelevato viene messa in atto l'istruzione sulla linea corrispondente come se aveste premuto «RETURN».

PROGRAMMAZIONE DEI TASTI FUNZIONE DEL C128

Come premete uno dei tasti funzione (da F1 a F8) il computer stampa un comando BASIC sullo schermo e in certi casi lo esegue immediatamente. Questi comandi sono conosciuti come i valori di default (standard) dei tasti funzione. Immettete un comando KEY senza parametri per ottenere un elenco delle definizioni dei tasti funzione.

CAMBIARE LE DEFINIZIONI DEI TASTI FUNZIONE

Potete cambiare le definizioni assegnate ad ogni tasto funzione immettendo il comando KEY seguito dal numero del tasto, poi una virgola, ed infine la nuova definizione in formato stringa. Per esempio:

```
KEY1,"DLOAD"+CHR$(34)+"NOME PROGRAMMA"
+CHR$(34)+CHR$(13)+"LIST"+CHR(13)
```

Quando premete F1 il computer caricherà il programma con quel nome e lo listerà non appena caricato. Il codice carattere delle virgolette è 34, ed è necessario per le operazioni di LOAD e SAVE. Il codice di ritorno a capo è 13 e serve per eseguire immediatamente il comando. Altrimenti le istruzioni saranno solo stampate sullo schermo e bisognerà premere il tasto «RETURN». Il seguente esempio usa il valore ASCII del tasto ESC per fare in modo che F3 esegua lo scroll in basso dello schermo:

```
KEY3,CHR$(27)+"W"
```

NOTA: Il totale delle definizioni degli 8 tasti funzione non deve eccedere i 246 caratteri.

COME USARE I VALORI DEI TASTI FUNZIONE DEL C64 IN MODO C128

Programmi scritti per il C64 che usano i valori dei tasti funzione possono essere usati lo stesso in Modo C128 assegnando prima i valori ASCII del C64 ai tasti funzione tramite la seguente linea:

```
10 J=132:FORA=1TO2:FORK=ATO8STEP2:J=J+1:KEYK,CHR$(J):NEXT:
   NEXT
```

COMPATTARE I PROGRAMMI BASIC

Parecchie tecniche conosciute generalmente come compattazione della memoria vi permettono di ottenere di più dalla memoria del vostro computer. Queste tecniche includono l'eliminazione degli spazi, l'uso di istruzioni multiple, lo sfruttamento della sintassi, la rimozione delle istruzioni REM, ed infine un uso più intelligente delle variabili, e del BASIC in generale.

ELIMINAZIONE DEGLI SPAZI

In molti comandi BASIC gli spazi non sono necessari, eccetto che tra virgolette, quando volete farli apparire sullo schermo. Sebbene gli spazi migliorino la leggibilità, gli spazi extra consumano memoria in più. Ecco una linea di istruzioni presentata in entrambi i modi:

```
10 INPUT"NOME";N$:FOR T=A TO M:PRINT "OK":
10 INPUT"NOME";N$:FORT=ATOM:PRINT"OK":
```

USO DI LINEE CON PIÙ ISTRUZIONI

I due punti (:) vi permettono di piazzare più istruzioni sulla stessa linea di programma. Ogni linea di programma consuma memoria in più. Fate attenzione, comunque, nel compattare le istruzioni IF. Ogni istruzione dopo IF, sulla stessa linea, può essere saltata insieme alla condizione IF...THEN. La linea seguente è equivalente alle cinque più sotto:

(A)
```
PRINTX:INPUTY:PRINTY:SCNCLR0:?J
```

(B)
```
10 PRINTX
20 INPUTY
30 PRINTY
40 SCNCLR0
50 ?J
```
Una linea sola richiede meno spazio in memoria e sul disco di cinque linee. L'esempio B richiede 16 byte in più, 2 per il numero di linea e 2 per il link.

SFRUTTAMENTO DELLA SINTASSI

La sintassi del BASIC è a volte molto flessibile e ciò può essere usato a vostro vantaggio. L'istruzione LET, per esempio, può essere scritta senza LET. LETY=2 è lo stesso che Y=2. Sebbene sia buona pratica inizializzare tutte le variabili a zero, non è indispensabile, poichè il computer le pone automaticamente a zero all'inizio del programma, vettori compresi. É necessario DIMensionare tutti i vettori per avere 12 o più elementi. Il C128 dimensiona automaticamente ogni vettore a 11 elementi se non viene specificata la dimensione in DIM. Spesso il punto e virgola (;) non è richiesto nelle istruzioni PRINT. Entrambe queste linee eseguono la stessa funzione:

```
10 PRINT"A";Z$;"WORD";CHR$(65);"NOW $"
20 PRINT"A"Z$;"WORD"CHR$(65)"NOW $"
```

RIMOZIONE DELLE ISTRUZIONI REM

Sebbene le istruzioni REM siano utili per il programmatore, la loro rimozione può liberare una considerevole quantità di memoria. Potrebbe essere una buona idea creare un listato separato con le righe REM.

USO DELLE VARIABILI

Rimpiazzate numeri ripetuti con una variabile. Questo è importante specialmente con grandi numeri come gli indirizzi di memoria. Usando POKE con parecchi numeri in sequenza si risparmia memoria se si usa una variabile, come POKE54273+V. Naturalmente, i nomi di variabile di una sola lettera richiedono il minimo della memoria. Riusate le vecchie variabili come quelle usate nei FOR...NEXT. Quando possibile, usate variabili intere, poichè consumano molto meno memoria di quelle reali.

USO INTELLIGENTE DEL BASIC

Se le informazioni sono usate ripetutamente, memorizzatele in vettori interi, se possibile. Usate linee DATA dove si può. Quando una linea simile è usata più volte, createne una sola con delle variabili ed accedetevi tramite GOSUB. Usate TAB e SPC invece di ampie stringhe di controllo del cursore.

MASCHERAMENTO DEI BIT

Ognuno dei bit componenti un byte può essere controllato individualmente usando gli operatori Booleani AND e OR. I calcoli con AND e OR sono basati su una tabella della verità (Tabella 3-1) mostrante i risultati delle possibili combinazioni degli argomenti.

X	Y	X AND Y	X OR Y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Tabella 3-1. Tabella della verità di AND e OR.

Con 0 rappresentante falso e 1 vero, la Tabella 3-1 mostra il funzionamento degli operatori AND e OR. Il risultato di un AND fra due bit è vero solo se entrambi sono 1, altrimenti è falso. Ogni combinazione con 0 dà uno 0 in un'operazione AND. Il risultato di un'operazione OR è falso solo se ogni bit è falso. Altrimenti il risultato è sempre vero. Ogni combinazione con un 1 dà sempre un 1. SOLO due zeri risultano in uno zero. Osservate l'esempio seguente con i numeri 5 e 6 in forma binaria. Quando digitate PRINT5AND6, il risultato è 4. Ecco perchè:

```
        5=  0000  0101
AND
        6=  0000  0110
     ------------------
        4=  0000  0100
```

Invece di sommare, AND esegue una comparazione bit a bit seguendo le regole della tabella della verità. Comparate colonna per colonna da destra: 1AND0=0, 0AND1=0, 1AND1=1, 0AND0=0. Il risultato "0100" convertito in decimale è 4.
Che effetto fa 5 OR 6? Compariamo di nuovo bit a bit, seguendo le regole della tabella della verità:

OR	**5=**	**0000**	**0101**
	6=	**0000**	**0110**
	7=	**0000**	**0111**

Il risultato 0111 è 7 in decimale. Notate da destra che 1OR0=1, 0OR1=1, 1OR1=1, e 0OR0=0. Conoscere il modo di funzionamento di AND e OR vi dà modo di controllare singoli bit della memoria del vostro computer. Molti byte utilizzano ogni bit per una funzione separata.

COME USARE AND E OR PER MODIFICARE SINGOLI BIT DI UN BYTE

Un byte è un gruppo di 8 cifre binarie etichettate, da destra a sinistra, da 0 a 7. Ogni posizione rappresenta un valore decimale uguale a 2 elevato il numero della posizione. Per esempio, la posizione 6 ha il valore decimale 2↑6, cioè 64. Da sinistra a destra abbiamo le posizioni:

7 6 5 4 3 2 1 0

i cui valori in decimale sono:

128 64 32 16 8 4 2 1

Per accendere un bit, mettete un 1 nella sua posizione. Per spegnerlo mettete invece uno 0. Perciò il valore binario 10010000 ha i bit 4 e 7 accesi. I loro valori sono 128 e 16. Così se voi inserite questo valore in un certo byte, solo quei due bit saranno accesi. Per accendere dei bit, memorizzate un valore corrispondente alla somma dei singoli valori dei bit accesi in quel byte. Naturalmente non sempre si conoscono i bit già accesi. Voi potete anche solo voler accendere dei bit specifici senza conoscere nè modificare il valore degli altri bit. Questo è lo scopo delle operazioni logiche AND e OR. Per prima cosa otteniamo il valore del byte tramite PEEK. Poi aggiungiamo il valore decimale del bit da accendere. Il comando seguente accende il bit 2 dell'indirizzo di memoria V:

```
POKEV,PEEK(V)+4
```

Con ciò si assume che il bit 2 (terzo da destra) abbia un valore di 0. Se fosse già acceso non si dovrebbe avere nessun effetto, mentre invece si ha un riporto dal bit 2 a quelli successivi. Per prevenire ciò il C128 usa la potenza della logica Booleana. Idealmente si vuole leggere (PEEK) ogni bit. L'approccio giusto è di eseguire un OR tra il byte ed un operando tale da ottenere il valore binario desiderato. Supponiamo di voler accendere il bit 5: l'operando sarà 00100000. Eseguire un OR tra questo valore ed un qualunque altro byte modificherà solo il bit 5, mettendolo sempre a 1 e lasciando gli altri invariati.

```
POKEV,PEEK(V)OR32
```

Proprio come OR accende un bit, AND può spegnerne uno - con una leggera differenza. AND risulta in un 1 solo se entrambi i bit confrontati sono a 1. Il

trucco sta nel confrontare il byte con un operando in cui tutti i bit sono a 1 eccetto quelli da spegnere. Per spegnere il bit 5, eseguite un AND fra il byte e l'immagine speculare dell'operando 00100000, cioè 11011111. In decimale questo operando è sempre 255 meno il valore del bit (o dei bit) da spegnere. Così:

```
POKEV,PEEK(V)AND(255-32)
```

spegne il bit 5.
Usate OR per accendere i bit.
Usate AND per spegnerli.

ESEMPI:

POKEW,PEEK(W)OR129	Accende i bit 0 e 7 dell'indirizzo W
POKES,PEEK(S)AND126	Spegne i bit 0 e 7 dell'indirizzo S (ricordate che 255-129=126)
POKEC,PEEK(C)AND254	Spegne il bit 0
POKEC,PEEK(V)OR63	Accende tutti i bit eccetto 6 e 7

MESSA A PUNTO DEI PROGRAMMI

Nessun programma è libero da errori alla prima scrittura. Il processo di trovarli e rimuoverli combina la redazione del programma con la risoluzione dei problemi.

ERRORI DI SINTASSI

Gli errori di sintassi risultano da errori di digitazione e di formato dei comandi BASIC. Viene stampato un messaggio di errore indicante la linea che lo ha generato. Digitando HELP «RETURN» o premendo il tasto HELP verrà listata ed evidenziata la linea errata. Errori di sintassi comuni sono termini BASIC mal digitati, punteggiatura scambiata, parentesi non aperte o chiuse, nomi di variabili riservate come TI, uso di numeri di linea inesistenti, ecc.

ERRORI DI LOGICA

Qualche volta c'è un errore nella logica del programma, cosicchè non vengono eseguiti i passi previsti. Alcuni errori sono causati dall'ordine delle istruzioni. Un errore comune è dimenticare che qualsiasi cosa sulla stessa linea dopo un'istruzione IF è condizionata dall'esito della IF.
Alcuni errori di logica richiedono un'investigazione a prova ed errore. La cosa migliore è farsi aiutare dal computer tramite il BASIC stesso.

L'USO DI UN RITARDO

Quando il computer risponde rapidamente, può aiutare inserire un ritardo tramite l'istruzione SLEEP in modo da capire cosa succede durante l'esecuzione.

L'USO DI PRINT E STOP

Inserite delle istruzioni STOP nel vostro programma prima della linea sospetta. Dei buoni posti sono alla fine di compiti specifici. Fate girare il programma. Dopo che l'istruzione STOP vi ha ridato il controllo, usate PRINT per trovare degli indizi sul problema, determinando i valori delle variabili, specialmente quelle interne ai cicli. Confrontateli con quelli che vi aspettate. Continuate poi con CONT fino alla prossima istruzione STOP.

COME INTRAPPOLARE UN ERRORE

Il 'debug' è l'arte di trovare la causa di un problema. Il programma seguente è del tutto valido, ma per B=0 dà un errore.

```
10 INPUTA,B
20 PRINTA/B
30 GOTO10
```

Sebbene in questo caso il computer definisca l'errore come una divisione per zero, non è sempre chiaro come B abbia assunto il valore zero. Potrebbe essere derivato da una formula insita nel programma oppure dando in input il valore zero alla variabile. Il comando TRAP intrappola gli errori di programma senza fermare l'esecuzione. Poichè non è possibile controllare per tutti i valori di B, potete intercettare un probabile errore di divisione per zero includendo un'istruzione TRAP all'inizio del programma.

```
5 TRAP50
10 INPUTA,B
20 PRINTA/B
30 GOTO10
50 PRINT"LA DIVISIONE PER ZERO É IMPOSSIBILE"
60 PRINT"IMMETTI UN ALTRO NUMERO PER B VICINO A ZERO"
70 RESUME
```

Bisogna usare RESUME dopo il trattamento dell'errore per riattivarne l'intrappolazione. Se includete l'opzione di immettere un altro valore per B, come segue:

```
65 INPUTB
```

allora RESUME senza numero di linea rieseguirà l'istruzione che ha dato l'errore. L'uso di RESUMENEXT fa riprendere l'esecuzione dall'istruzione seguente, in questo caso la linea 30. TRAP dice al computer di andare al numero di linea specificato nel caso di errore. NON usate TRAP finchè non avete eliminato tutti gli errori di sintassi. TRAP può trovare la condizione di errore solo mentre la sta cercando. Un errore di logica o di sintassi nella routine di gestione dell'errore può causare un errore o impedire l'individuazione di un altro. In altre parole, le routine di TRAP non devono generare un errore al loro interno.

LE FUNZIONI DI ERRORE

Parecchie variabili riservate connesse al sistema memorizzano informazioni riguardanti gli errori di programma. ER conserva il numero dell'errore. EL il numero di linea dove si è verificato. Nell'esempio della divisione per zero, ERR(ER) ritorna "DIVISION BY ZERO" e EL il numero di linea dell'errore, cioè 20. Aggiungete queste funzioni al programma precedente. Vedete l'Appendice A per un elenco completo degli errori.

GLI ERRORI DEL DOS

Le informazioni sugli errori da disco provengono dalle variabili DS e DS$, dove DS è il numero dell'errore (guardare nell'Appendice B) e DS$ fornisce il numero dell'errore, il messaggio e la traccia e settore che lo hanno generato. DS legge il canale degli errori del disco e si usa durante un'operazione su disco per sapere perchè la luce lampeggia. Provare a leggere una Directory senza il dischetto dentro il drive origina l'errore seguente, dato da PRINT DS:

```
74,DRIVE NOT READY,00,00
```

L'Appendice B evidenzia le cause di ogni errore. Per fare in modo che un tasto funzione legga automaticamente il canale degli errori usate:

```
KEY1,"PRINT DS$+CHR$(13)
```

COME RINTRACCIARE UN ERRORE

Alcuni programmi hanno molti cicli complessi noiosi da seguire. Un tracciamento metodico passo passo può essere utile. I comandi TRON e TROFF possono essere usati entro un programma come uno strumento di messa a punto per seguire certe routine.
Alcuni errori possono essere trovati agendo solo come il computer e seguendo metodicamente ogni istruzione passo passo, ed infine facendo tutti i calcoli finchè non si scopre qualcosa di sbagliato. Fortunatamente il Commodore 128 può seguire gli errori al posto vostro. Immettete il comando TRON prima di eseguire un programma. Il programma mostra ogni numero di linea eseguito tra parentesi quadre, seguito dai risultati del programma (per rallentare la visualizzazione premete il tasto Commodore (C=)).
Provatelo con questo doppio ciclo:

```
10 FORA=1T05
20 FORB=2T06
30 C=B*A:K=K+C:PRINTK
40 NEXTB:NEXTA
50 PRINTK
```

I risultati saranno come questi:

```
[10][20][30][30][30]2
[40][30][30][30]5
```

e significano che il primo risultato stampato è il numero 2 dopo aver eseguito le linee 10, 20, 30. Poi l'esecuzione delle linee 40 e 30 darà 5, ecc. Notate che nella linea 30 vengono eseguite tre istruzioni. Per disattivare la funzione di tracciamento usate TROFF.

FINESTRAZIONE

La grandezza standard dello schermo è di 40-80 colonne per 25 righe. Spesso è conveniente avere disponibile solo una porzione dello schermo per qualche altro lavoro. Il procedimento per produrre ed isolare piccole aree dello schermo è chiamato "finestrazione".

DEFINIZIONE DI UNA FINESTRA

Ci sono due modi di creare una finestra:
il primo direttamente, il secondo da programma tramite il comando WINDOW.
Nel primo, tutto quanto occorre per descrivere e creare una finestra è usare il tasto ESC seguito da T o B.
Ecco come definire una finestra in modo diretto:

1. Muovete il cursore all'angolo superiore sinistro della finestra voluta. Premete il tasto ESC e poi il tasto T.
2. Muovete il cursore all'angolo inferiore destro e premete il tasto ESC e poi il tasto B.

Ora la vostra finestra è in azione, tutti i comandi e i listati rimarranno nella finestra finchè non premerete due volte il tasto HOME. Ciò è utile se avete un listato sullo schermo principale e volete mantenerlo mentre stampate qualcos'altro in una finestra. Guardate al Capitolo 13, il Sistema Operativo del Commodore 128, nella sezione dell'editor di schermo per i controlli speciali del tasto ESC in una finestra.
Nonostante sia possibile definire numerose finestre simultaneamente sullo schermo, solo una può essere usata in un dato istante. Le altre finestre rimangono sullo schermo, ma sono inattive. Per rientrare in una finestra da cui eravate usciti ridefinite l'angolo superiore sinistro e quello inferiore destro coi comandi ESC T e ESC B come avevate già fatto.
Il secondo modo di definire una finestra è tramite il comando BASIC di finestrazione. Il comando:

```
WINDOW20,12,39,24,1
```

stabilisce una finestra col margine superiore sinistro alla riga 12, colonna 20, e il margine inferiore destro alla colonna 39, riga 24. L'1 significa che l'area deve essere cancellata. Una volta dato questo comando, tutte le attività si manterranno in quest'area.
Usate il comando WINDOW da programma quando volete eseguire un'attività in un'area ristretta dello schermo.

TECNICHE DI PROGRAMMAZIONE BASIC AVANZATA PER I MODEM COMMODORE

Le seguenti informazioni vi dicono come:

1. Generare le frequenze Touch Tone
2. Rilevare lo squillo del telefono
3. Programmare il telefono per prendere o rilasciare la linea
4. Rilevare la portante

Le procedure di programmazione funzionano in Modo C128 o C64 con il Modem/300. Nel Modo C128 selezionate una configurazione di banco con BASIC, KERNAL e I/O.

COME GENERARE LE FREQUENZE DI TOUCH TONE (DTMF)

Ogni tasto di un telefono Touch Tone genera una differente coppia di toni (frequenze). Potete simulare questi toni con il vostro Commodore 128. Ogni bottone ha un valore di riga e di colonna al quale corrisponde una determinata locazione di memoria per ottenere la frequenza corretta. Ecco i valori di frequenza di ogni riga e colonna relativi ad ogni tasto del telefono Touch Tone:

TAVOLA DI FREQUENZA TOUCH TONE

Righe:	**Colonne:**		
	1 1029Hz	**2 1336Hz**	**3 1477Hz**
1 697Hz	**1**	**2**	**3**
2 770Hz	**4**	**5**	**6**
3 852Hz	**7**	**8**	**9**
4 941Hz	*****	**0**	#

Per generare questi toni in BASIC col vostro Commodore 128 seguite questa procedura:

1. Inizializzate il chip del suono (SID) con le seguenti istruzioni:

```
SID=54272
POKESID+24,15:POKESID+4,16
POKESID+11,16:POKESID+5,0:POKESID+12,0
POKESID+6,15*16:POKESID+13,15*16:POKESID+23,0
```

2. Poi, selezionate un valore di riga e colonna per ogni cifra del numero telefonico. Le istruzioni POKE per ogni riga e colonna sono come segue:

```
Colonna 1: POKESID,117:POKESID+1,77
Colonna 2: POKESID,152:POKESID+1,85
Colonna 3: POKESID,161:POKESID+1,94
Riga 1: POKESID+7,168:POKESID+8,44
Riga 2: POKESID+7,85:POKESID+8,49
Riga 3: POKESID+7,150:POKESID+8,54
Riga 4: POKESID+7,74:POKESID+8,60
```

Per esempio, per generare il tono corrispondente al numero 1, inserite i valori per la riga 1, colonna 1 come segue:

```
POKESID+7,168:POKESID+8,44:REM RIGA 1
POKESID,117:POKESID+1,77:REM COLONNA 1
```

3. Accendete le voci e aggiungete un ritardo con queste istruzioni:

```
POKESID+4,17::POKESID+11,17:REM ABILITA LE VOCI
FORI=1TO50:NEXT:REM RITARDO
```

4. Spegnete le voci e aggiungete un ritardo:

```
POKESID+4,16:POKESID+11,16:REM DISABILITA LE VOCI
FORI=1TO50:NEXT:REM RITARDO
```

5. Ora ripetete i passi dal 2 al 4 per ogni cifra del numero da comporre.
6. Infine, disabilitate il chip SID con questa istruzione:

```
POKESID+24,0
```

COME RILEVARE LO SQUILLO DEL TELEFONO

Per sapere, col vostro Commodore 128, se il telefono sta suonando, usate questa istruzione:

```
IF(PEEK(56577)AND8)=0THENPRINT"STA SUONANDO"
```

Se il bit 3 di 56577 è diverso da 0, cioè 1, allora il telefono non sta suonando.

COME PROGRAMMARE IL TELEFONO PER PRENDERE O RILASCIARE LA LINEA

Per programmare il telefono in modo da prendere la linea usate le seguenti istruzioni in un programma:

```
OH=56577:HI=32:LO=255-32
POKE(OH+2),(PEEK(OH+2)ORHI)
POKEOH,(PEEK(OH)ANDLO)
```

Per riappendere il telefono usate quest'istruzione:

```
POKEOH,(PEEK(OH)ORHI)
```

Ecco la procedura per entrare in comunicazione con una chiamata:

1. Porre il modem in Originate
2. Programmare il telefono per prendere la linea
3. Aspettare 2 secondi (FORI=1TO500:NEXT:REM RITARDO DI 2 SECONDI)
4. Comporre ogni cifra e farla seguire da un ritardo (FORI=1TO50:NEXT)
5. Quando viene rilevata la portante (un tono alto), il Modem/300 si mette automaticamente in linea col computer che ha risposto
6. Programmare il telefono per rilasciare la linea quando avete finito.

Ecco invece la procedura per rispondere ad una chiamata:

1. Porre il modem in Answer
2. Per rispondere manualmente, programmate il telefono per prendere la linea
3. Per rispondere automaticamente, rilevate se il telefono sta suonando e poi fategli prendere la linea
4. Ora il Modem/300 risponde automaticamente alla chiamata
5. Fate rilasciare la linea.

RILEVAMENTO DELLA PORTANTE

I Modem/1200 e Modem/300 Commodore sono costruiti con la capacità di rilevare la portante.
Questa capacità è utile per un inaspettato modo di auto-risposta (auto-answer). Controllando la linea della portante, il computer può essere programmato per rilasciare la linea dopo una perdita della portante. Poichè il chiamante può dimenticare di rilasciare la linea, il vostro programma dovrebbe controllare le linee di trasmissione e ricezione dati. Nel caso restassero inattive per 5 minuti o più, il modem stesso dovrebbe rilasciare la linea. Per rilevare la portante col vostro computer, usate le seguenti istruzioni:

```
OH=56577
IF((PEEK(OH)AND16)=0)THENPRINT"RILEVATA LA PORTANTE"
```

Se il bit 4 di 56577 ha un valore diverso da 0, cioè 1, allora non c'è portante.

COMPOSIZIONE DEL NUMERO AD IMPULSI

Per comporre un numero con un modem, il programma del computer deve generare degli impulsi entro un tempo prestabilito. Negli Stati Uniti e nel Canada, il tempo è tra 8 e 10 impulsi al secondo con un ciclo di ritardo fra ogni impulso variante fra il 58% ed il 64%. Molte persone, comunque, usano 10 impulsi al secondo con un ciclo di ritardo del 60%.

Così per effettuare una chiamata, il vostro programma deve prendere la linea (lo stesso che sollevare la cornetta del telefono). Poi per comporre la prima cifra, un 3 per esempio, il programma deve rilasciare la linea per 60 millisecondi, poi riprenderla per 40. Ciò va ripetuto 3 volte per indicare la cifra 3.
Una volta composta una cifra, fate una pausa di almeno 600 millisecondi, poi ripete il procedimento per tutte le altre cifre. Nel caso dello 0, dovrete emettere 10 impulsi.

USO DEI CODICI ESC

Per usare le opzioni di ESC da programma, usate una linea come:

```
10150 PRINTCHR$(27)"U"
```

per avere un cursore lineetta (solo a 80 colonne). Per esempio, per cancellare lo schermo dal cursore fino in fondo alla finestra:

```
10160 PRINTCHR$(27)"@"
```

(Andate alla sezione dell'Editor di Schermo del Capitolo 13 per vedere tutti i codici ESC e di controllo del Commodore 128).

RILOCAMENTO DEL BASIC

Per rilocare l'inizio o la fine del programma BASIC (in Modo C128) per avere memoria in più o proteggere programmi in linguaggio macchina dalla sovrascrittura del testo BASIC, è necessario ridefinire i puntatori di inizio e fine a specifici indirizzi di memoria.
L'inizio del testo BASIC è nel puntatore in 45-46 ($2D-$2E). La fine del testo è invece in 4626-4627 ($1212-$1213). Le seguenti istruzioni mostrano rispettivamente l'inizio e la fine del testo BASIC (se la pagina grafica non è allocata):

```
PRINTPEEK(45),PEEK(46),PEEK(4626),PEEK(4627)
```

1 28 0 255

Poichè il secondo numero di ogni indirizzo è il byte alto, l'inizio standard del basic è 28*256+1 o 7169 e la fine 255*256+0 o 65280.
Il seguente comando riduce l'area del basic di 4 Kbyte abbassando la fine del testo a 61184 (239*256):

```
POKE4626,0:POKE4627,239:NEW
```

Per muovere l'inizio del BASIC più su di 1 Kbyte, da 7168 a 8192, fate:

```
POKE46,32:POKE45,1:NEW
```

Questo è il caso in cui non è stato allocato uno schermo grafico. Ricordate, l'inizio del BASIC è a 16384 ($4000) quando allocate una pagina grafica, ed il programma viene spostato più in alto.

COME UNIRE PROGRAMMI E FILE

I file possono essere uniti aprendo un file esistente e mettendo il puntatore alla fine del file, in modo da scrivere i nuovi dati in coda al file. Nel BASIC del C128 è stato incluso il comando APPEND per questo compito:

APPEND#5,''nome del file''

apre il canale 5 verso un file esistente di nome ''nome del file''. Le istruzioni di scrittura successive (PRINT#5) aggiungeranno nuovi dati al file. APPEND si usa per lo più con i file di dati.
Il comando CONCAT permette la concatenazione (unione in sequenza) di due file o programmi mantenendo il nome del primo.

CONCAT''parte2b''TO''parte2''

crea un nuovo file chiamato ''parte2'', consistente della vecchia ''parte2'' più la nuova ''parte2b'' in sequenza. Programmi BASIC concatenati devono solo essere rinumerati per funzionare correttamente. Altre correzioni non sono necessarie.
Le routine BASIC descritte in questo capitolo possono aumentare grandemente le capacità dei vostri programmi. A questo punto, il BASIC è stato discusso dettagliatamente. La programmazione in linguaggio macchina introdotta nel Capitolo 6 può estendere anche oltre le capacità dei programmi. E, come mostrato nel Capitolo 8 (vol. III), per avere una ancor maggiore flessibilità e potenza, potete combinare il BASIC ed il linguaggio macchina nei vostri programmi.

5

PROGRAMMAZIONE GRAFICA DEL COMMODORE 128

COME USARE IL SISTEMA GRAFICO

CARATTERISTICHE VIDEO DEL COMMODORE 128

In Modo C128, il BASIC Commodore 7.0 offre 14 comandi grafici ad alto livello che rendono facili i lavori di programmazione complessi. Potete ora tracciare cerchi, quadrati, linee, punti ed altre forme geometriche con 10 comandi ad alto livello del tipo di DRAW, BOX e CIRCLE, e con 4 comandi per gli sprite (i comandi relativi agli sprite sono descritti nel Capitolo 10 [Vol. IV]). Non dovete più essere un programmatore in linguaggio macchina o comperare pacchetti grafici addizionali per mostrare intricati e piacevoli disegni - le capacità grafiche del BASIC del Commodore 128 pensano a questo al vostro posto. Naturalmente, se siete un programmatore in linguaggio macchina o uno sviluppatore di software le eccezionali caratteristiche hardware del video del C128 offrono un alto rapporto prezzo/prestazioni per ogni applicazione su microcomputer.

Le caratteristiche grafiche del C128 comprendono:

- Comandi grafici e di sprite
- 16 colori
- 6 modi di visualizzazione:
  Modo carattere standard
  Modo carattere multicolore
  Modo colore di fondo esteso
  Modo bit map standard
  Modo bit map multicolore
  Modi carattere e bit map combinati (schermi divisi)
- 8 SPRITE liberamente definibili e spostabili coi quali è possibile effettuare un'animazione
- caratteri ridefinibili dall'utente
- scroll fine orizzontale e verticale

Il Commodore 128 può produrre due tipi di segnali video: video composito a 40 colonne, e video RGBI a 80 colonne. Il segnale video composito, incanalato attraverso un chip (8564) VIC II - simile a quello usato sul C64 - miscela tutti i colori dello spettro in un segnale singolo inviato al monitor. L'8563 separa i colori rosso, verde e blù per pilotare separatamente i cannoni a raggi catodici del monitor ottenendo un'immagine più pulita, incisiva e netta rispetto al video composito.
Il chip VIC II supporta tutti i comandi grafici del BASIC Commodore 7.0, gli SPRITE, 16 colori e i modi di visualizzazione sopra menzionati. Il chip a 80 colonne, disegnato principalmente per le applicazioni commerciali, supporta anch'esso 16 colori (alcuni dei quali differenti da quelli del VIC II), modo testo e bit map standard. Non sono disponibili gli sprite. Il BASIC Commodore 7.0 non supporta il modo bit map a 80 colonne, che può comunque essere usato tramite

programmi in linguaggio macchina. Guardate al Capitolo 11 (Vol. IV), Programmazione del chip a 80 colonne (8563), per informazioni sul modo bit map a 80 colonne.
Questo capitolo verte sull'uso delle caratteristiche grafiche del Commodore 128 attraverso il BASIC ed usando lo schermo a 40 colonne. Eccettuati i comandi di sprite, ogni comando grafico è elencato in ordine alfabetico. I comandi di sprite sono trattati nel Capitolo 10 (Vol. IV). Di seguito al formato di ogni comando vi sono programmi esemplificativi illustranti le caratteristiche di ogni comando. Dove possibile, sono state incluse routine in linguaggio macchina equivalenti al comando grafico per illustrare il modo di operare del linguaggio macchina.
Il Capitolo 9 (Vol. III), La potenza dietro la grafica del Commodore 128, è una descrizione del funzionamento interno delle capacità grafiche del Commodore 128. Esso spiega l'uso della memoria di schermo, colore e carattere ed il modo di leggervi e scrivervi un dato. Il Capitolo 10 (Vol. IV) poi spiega come usare gli sprite coi nuovi comandi BASIC. Lo stesso capitolo tratta anche del funzionamento interno degli sprite, la memorizzazione, l'indirizzamento, gli assegnamenti del colore, ed infine il controllo tramite il linguaggio macchina.

TIPI DI VISUALIZZAZIONE

Il vostro C128 visualizza le informazioni in parecchi modi differenti; il parametro "sorgente" del comando si riferisce a tre differenti modi di schermo.

SCHERMO DI TESTO

Lo schermo di testo mostra solo testo o caratteri, come lettere, numeri, simboli speciali ed i caratteri grafici raffigurati su molti tasti del C128. Il C128 può mostrare del testo sia a 40 che a 80 colonne. Lo schermo di testo include il modo carattere standard, modo carattere multicolore e modo colore di fondo esteso.
Il Commodore 128 normalmente opera nel modo carattere standard. All'accensione siete automaticamente in modo carattere standard, così come anche quando scrivete programmi. Il modo carattere standard visualizza caratteri in uno di 16 colori su uno sfondo in uno tra 16 colori.
Il modo multicolore vi dà un maggior controllo sul colore del modo standard. Ogni punto dello schermo, un pixel, in una griglia di 8 per 8 può avere uno fra 4 colori, in confronto al modo standard che può avere solo uno fra 2 colori. Il modo multicolore usa due registri addizionali del colore di fondo. I tre registri del colore di fondo più la memoria colore di ogni carattere vi danno la possibilità di scegliere 4 colori per ogni punto in una griglia di 8 per 8.
Ogni pixel in modo multicolore è doppio in larghezza rispetto ad un pixel dei modi carattere e bit map standard. Ne risulta una risoluzione orizzontale multicolore dimezzata rispetto al normale (160x200). Comunque, il maggior controllo sul colore compensa di gran lunga la risoluzione orizzontale dimezzata.
Il modo colore esteso vi permette di controllare il colore di fondo e di primo piano di ogni carattere. Il modo colore esteso usa tutti e quattro i registri del colore di fondo, ma limita il set di caratteri ai primi 64. Il secondo set, infatti, è identico al primo, ma usa un diverso colore di fondo, e così per i due set rimanenti. Il colore

del carattere è poi controllato dalla memoria colore. Per esempio, in modo colore esteso, potete visualizzare un carattere porpora su uno sfondo verde ed uno schermo nero.
Ognuno dei modi di visualizzazione di testo riceve le informazioni sui caratteri da uno di due posti della memoria. Normalmente, le informazioni sui caratteri sono prese da una memoria su ROM (Read Only Memory), ma il C128 vi permette di disegnare un vostro set di caratteri e sostituirlo all'originale. Chiaramente il vostro set andrà su RAM.

SCHERMO GRAFICO (BIT MAP)

Il modo bit map vi permette di visualizzare grafica dettagliata, disegni ed intricati tracciamenti. Questo tipo di schermo comprende il modo bit map standard e bit map multicolore. Il modo bit map vi permette di controllare ogni singolo pixel (picture element, elemento del disegno) ottenendo così un elevato dettaglio per disegni ed altra computer art. Questi modi grafici sono supportati dal BASIC solo per il chip VIC II.
Il chip a 80 colonne è stato progettato primariamente per la visualizzazione di caratteri, ma potete usarlo graficamente tramite i vostri programmi. Guardate al Capitolo 11 (Vol. IV), Programmazione del chip a 80 colonne (8563), per informazioni più dettagliate.
La differenza fra il testo e i modi bit map sta nel modo di memorizzare e reperire le informazioni. Lo schermo di testo può manipolare solo interi caratteri, ognuno dei quali copre un'area di 8 per 8 pixel. Il più potente modo bit map controlla ogni singolo punto.
Il modo bit map standard vi permette di assegnare ad ogni pixel uno fra due colori. Il 'bit mapping' è una tecnica per la quale ogni punto sullo schermo è rappresentato da un bit in memoria. Nel modo standard, un punto spento equivale a un bit a zero in memoria. Se invece il punto è acceso, cioè del colore di primo piano, allora il bit in memoria è a uno. La serie di 64000 pixel equivale a 64000 bit in memoria, che controllano l'immagine che vedete sullo schermo. Molta della dettagliatissima grafica al computer che vedete in dimostrazioni e videogiochi è una grafica bit map.
Il modo bit map multicolore è una combinazione dei modi bit map standard e carattere multicolore. Potete mostrare ogni punto dello schermo in uno dei 4 colori entro una griglia di 8 per 8 punti. Di nuovo, come nel modo carattere multicolore, c'è uno scambio fra risoluzione orizzontale e controllo del colore.

SCHERMO MISTO

Il terzo tipo di schermo, schermo misto, è una combinazione dei primi due tipi. Lo schermo misto mostra parte dello schermo in modo testo e parte in modo bit map (sia standard che multicolore). Il C128 è capace di ciò in quanto usa due parti della sua memoria per i due schermi: una parte per il testo e l'altra per la grafica.

SOMMARIO DEI COMANDI

Fa seguito una breve spiegazione di ogni comando grafico disponibile nel BASIC 7.0

BOX:	Traccia rettangoli sullo schermo grafico
CHAR:	Mostra caratteri sullo schermo grafico e di testo
CIRCLE:	Traccia cerchi, ellissi ed altre forme geometriche
COLOR:	Seleziona i colori di bordo, primo piano, fondo e caratteri
DRAW:	Traccia linee e punti
GRAPHIC:	Seleziona uno schermo (testo, bit map e diviso)
GSHAPE:	Estrae dati da una variabile stringa e li traccia in una specifica posizione dello schermo
LOCATE:	Posiziona il cursore grafico sullo schermo
PAINT:	Riempie con un colore un area dello schermo grafico
SCALE:	Imposta la grandezza relativa dello schermo grafico
SSHAPE:	Salva una parte dello schermo grafico in una variabile stringa
WIDTH:	Imposta la larghezza dei punti tracciati

I paragrafi seguenti danno formato ed esempi per ognuno dei 12 comandi grafici del BASIC 7.0 non relativi agli sprite. Per una spiegazione completa di ognuno riferitevi all'Enciclopedia del BASIC 7.0 nel Capitolo 2.

BOX

Traccia un rettangolo in una posizione specificata

BOX[sorg col] , X1, Y1[,X2,Y2][,angolo][,riempi]

dove:

sorgente colore	0=colore di fondo 1=primo piano 2=multi 1 3=multi 2
X1,Y1	angolo sup. sinistro
X2,Y2	angolo inf. destro (default il cursore pixel)
angolo	rotazione in gradi (default 0)
riempi	riempi la forma: 0=non riempire 1=riempi

ESEMPI

```
10 COLOR0,1:COLOR1,6:COLOR4,1
20 GRAPHIC1,1:REM SELEZIONA BIT MAP
30 BOX1,10,10,70,70,90,1:REM TRACCIA IN VERDE UN QUADRATO PIENO
40 FORI=20TO140STEP3
50 BOX1,I,I,I+60,I+60,I+80:REM TRACCIA RETTANGOLI RUOTATI
60 NEXT
70 BOX1,140,140,200,200,220,1:REM TRACCIA IN VERDE UN SECONDO QUADRATO PIENO
80 COLOR1,3:REM DA VERDE A ROSSO
90 BOX1,150,20,210,80,90,1:REM TRACCIA IN ROSSO UN RETTANGOLO PIENO
100 FORI=20TO140STEP3
110 BOX1,I+130,I,I+190,I+60,I+70:REM TRACCIA RETTANGOLI ROSSI RUOTATI
120 NEXT
130 BOX1,270,140,330,200,210,1:REM TRACCIA IN ROSSO UN SECONDO RETTANGOLO PIENO
140 SLEEP5:REM RITARDO
150 GRAPHIC0,1:REM RITORNO IN MODO TESTO
```

```
10 COLOR0,1:COLOR4,1:COLOR1,6
20 GRAPHIC1,1
30 BOX1,0,0,319,199
40 FORX=10TO160STEP10
50 C=X/10
60 COLOR1,C
70 BOX1,X,X,320-X,320-X
80 NEXT
90 SLEEP5
100 GRAPHIC0,1
```

```
10 COLOR0,1:COLOR4,1:COLOR1,6
20 GRAPHIC1,1
30 BOX1,50,50,150,120
40 BOX1,70,70,170,140
50 DRAW1,50,50TO70,70
60 DRAW1,150,120TO170,140
70 DRAW1,50,120TO70,140
80 DRAW1,150,50TO170,70
90 CHAR1,20,20,"ESEMPIO DI CUBO"
100 SLEEP5
110 GRAPHIC0,1
```

```
10 COLOR1,6:COLOR4,1:COLOR0,1
20 GRAPHIC1,1:REM SELEZIONA IL MODO BIT MAP
30 DO:REM CALCOLA DEI PUNTI CASUALI
40 X1=INT(RND(1)*319+1)
50 X2=INT(RND(1)*319+1)
60 X3=INT(RND(1)*319+1)
70 X4=INT(RND(1)*319+1)
80 Y1=INT(RND(1)*199+1)
90 Y2=INT(RND(1)*199+1)
100 Y3=INT(RND(1)*199+1)
110 Y4=INT(RND(1)*199+1)
120 BOX1,X1,Y1,X2,Y2:REM TRACCIA DEI RETTANGOLI CASUALI
130 BOX1,X3,Y3,X4,Y4
140 DRAW1,X1,Y1TOX3,Y3:REM CONGIUNGE I PUNTI
150 DRAW1,X2,Y2TOX4,Y4
160 DRAW1,X1,Y2TOX3,Y4
170 DRAW1,X2,Y1TOX4,Y3
180 SLEEP2:REM RITARDO
190 SCNCLR
200 LOOP:REM CICLO INFINITO
```

CHAR

Stampa caratteri in una specifica posizione

CHAR[sorg col],X,Y[,stringa][,RVS]

Quest'istruzione è usata principalmente per stampare caratteri su uno schermo bit map, ma si può usare anche in modo testo. Ecco il significato dei vari parametri:

sorg col	0=sfondo 1=primo piano
X	colonna (0-79: a 40 colonne va a capo)
Y	riga (0-24)
stringa	stringa da stampare
RVS	flag di reverse (0=no, 1=sì)

ESEMPIO:

```
10 COLOR2,3:REM MULTI 1=ROSSO
20 COLOR3,7:REM MULTI 2=BLÙ
30 GRAPHIC3,1
40 CHAR0,10,10,"TESTO",0
```

CIRCLE

Traccia cerchi, ellissi, archi, ecc. a posizioni specificate

CIRCLE[sorg col],X,Y[,rX][,rY][,ai][,af][,angolo][,inc]

dove:

sorg col	0=sfondo 1=primo piano 2=multi 1 3=multi 2
X,Y	coordinate del centro
rX	raggio X (scalato)
rY	raggio Y (default rX)
ai	angolo iniziale (default 0 gradi)
af	angolo finale (default 360 gradi)
angolo	rotazione (default 0 gradi)
inc	incremento (default 2 gradi)

ESEMPI:

CIRCLE1,160,100,65,10	Traccia un'ellisse
CIRCLE1,160,100,65,65	Traccia un cerchio
CIRCLE1,60,40,20,18,,,,45	Traccia un ottagono

CIRCLE1,260,40,20,,,,,90	Traccia un diamante
CIRCLE1,60,140,20,18,,,,120	Traccia un triangolo
CIRCLE1,+2,+2,50,50	Traccia un cerchio 2 pixel a destra e in basso della precedente posizione

PROGRAMMI DI ESEMPIO:

```
10 REM SISTEMA SOTTOMARINO DI TRACCIAMENTO
20 COLOR0,1:COLOR4,1:COLOR1,2:REM SELEZIONA FOND, BORDO E PRIMO PIANO
30 GRAPHIC1,1:REM ENTRA IN MODO BIT MAP
40 BOX1,0,0,319,199
50 CHAR1,2,24,"SISTEMA SOTTOMARINO DI TRACCIAMENTO":REM STAMPA DEI CARATTERI IN PAGINA GRAFICA
60 COLOR1,3:REM SELEZIONA IL ROSSO
70 XR=0:YR=0:REM INIZIALIZZA I RAGGI X E Y
80 DO
90 CIRCLE1,160,100,XR,YR,0,360,0,2:REM TRACCIA DEI CERCHI
100 XR=XR+10:YR=YR+10:REM AGGIORNA I RAGGI
110 LOOPUNTILXR=90
120 DO
130 XR=0:YR=0
140 DO
150 CIRCLE0,160,100,XR,YR,0,360,0,2:REM CANCELLA IL CERCHIO
160 COLOR1,2:REM PASSA IN BIANCO
170 DRAW1,160,100+XR:DRAW0,160,100+XR:REM TRACCIA IL BLIP SOTTOMARINO
180 COLOR1,3:REM TORNA IN ROSSO
190 SOUND1,16000,15:REM BLIP
200 CIRCLE1,160,100,XR,YR,0,360,0,2:REM TRACCIA IL CERCHIO
210 XR=XR+10:YR=YR+10:REM AGGIORNA I RAGGI
220 LOOPUNTILXR=90:REM CICLO
230 LOOP
```

```
10 COLOR0,1:COLOR4,1:COLOR1,7
20 GRAPHIC1,1:REM SELEZIONA LA BIT MAP
30 X=150:Y=150:XR=150:YR=150
40 DO
50 CIRCLE1,X,Y,XR,YR
60 X=X+7:Y=Y-5:REM INCREMENTA LE COORDINATE X E Y
70 XR=XR-5:YR=YR-5:REM DECREMENTA I RAGGI X E Y
80 LOOPUNTILXR=0
90 GRAPHIC0,1:REM SELEZIONA IL TESTO
```

COLOR

Definisce i colori di ogni area di schermo

COLOR num sorg,num col

Quest'istruzione assegna un colore ad una delle seguenti sette aree:

AREA	SORGENTE
0	**fondo a 40 colonne**
1	**primo piano a 40 colonne**
2	**multi 1**
3	**multi 2**
4	**bordo a 40 colonne**
5	**colore carattere**
6	**fondo a 80 colonne**

I codici colore vanno da 1 a 16.

CODICE	COLORE	CODICE	COLORE
1	nero	9	arancio
2	bianco	10	marrone
3	rosso	11	rosso chiaro
4	ciano	12	grigio scuro
5	porpora	13	grigio medio
6	verde	14	verde chiaro
7	blu	15	azzurro
8	giallo	16	grigio chiaro

Codici dei colori a 40 colonne

ESEMPIO:

COLOR0,1 colore di fondo a 40 colonne nero
COLOR5,8 colore carattere giallo

PROGRAMMA DI ESEMPIO:

```
10 REM CAMBIA IL COLORE BIT MAP DI PRIMO PIANO
20 GRAPHIC1,1
30 I=1
40 DO
50 COLOR1,I
60 BOX1,100,100,219,159
70 I=I+1:SLEEP1
80 LOOPUNTILI=17
90 GRAPHIC0,1
100 REM CAMBIA IL COLORE DEL BORDO
110 I=1
120 DO
130 COLOR4,I
140 I=I+1:SLEEP1
150 LOOPUNTILI=17
160 REM CAMBIA IL COLORE DEI CARATTERI
170 I=1
180 DO
190 COLOR5,I
200 PRINT"CODICE COLORE"I
210 I=I+1:SLEEP1
220 LOOPUNTILI=17
230 REM CAMBIA IL COLORE DI FONDO
240 I=1
250 DO
260 COLOR0,I
270 I=I+1:SLEEP1
280 LOOPUNTILI=17
290 COLOR0,1:COLOR4,1:COLOR5,2
```

DRAW

Traccia punti, linee e forme alla posizione specificata

DRAW[sorg col],[X,Y][TOX2,Y2]...

Ecco i valori dei parametri:

sorg col	0=fondo
	1=primo piano
	2=multi 1
	3=multi 2
X,Y	coordinate di partenza
X2,Y2	coordinate di arrivo

ESEMPI:

DRAW1,100,50	Traccia un punto
DRAW,10,10TO100,60	Traccia una linea
DRAW,10,10TO10,60TO100,60TO10,10	Traccia un triangolo

PROGRAMMI DI ESEMPIO:

```
10 REM ESEMPI DI TRACCIAMENTO
20 COLOR0,1:COLOR4,1:COLOR1,6
30 GRAPHIC1,1
40 CHAR1,12,1,"IL COMANDO DRAW"
50 X=10
60 DO
70 DRAW1,X,50:REM TRACCIAMENTO PUNTI
80 X=X+10
90 LOOPUNTILX=320
100 CHAR1,11,7,"TRACCIA DEI PUNTI"
110 Y=70
120 DO
130 Y=Y+5
140 DRAW1,1,YTOY,Y:REM TRACCIAMENTO LINEE
150 LOOPUNTILY=130
160 CHAR1,18,11,"LINEE"
170 DRAW1,10,140TO10,199TO90,165TO40,160TO10,140:REM TRACCIA LA FORMA 1
180 DRAW1,120,145TO140,195TO195,195TO225,145TO120,145:REM TRACCIA LA FORMA 2
190 DRAW1,250,199TO319,199TO319,60TO250,199:REM TRACCIA LA FORMA 3
200 CHAR1,22,15,"E FORME"
210 SLEEP5:GRAPHIC0,1
```

```
10 COLOR0,1:COLOR4,1:COLOR1,7
20 GRAPHIC1,1:REM SELEZIONA LA BIT MAP
30 Y=1
40 DO
50 DRAW1,1,YTO320,Y:REM TRACCIA DELLE LINEE ORIZZONTALI
60 Y=Y+10
70 LOOPWHILEY<200
75 X=1
80 DO
90 DRAW1,X,1TOX,200:REM TRACCIA DELLE LINEE VERTICALI
95 X=X+10
97 LOOPWHILEX<320
100 COLOR1,3:REM PASSA IN ROSSO
110 DRAW1,0,100TO320,100:REM TRACCIA L´ASSE X IN ROSSO
120 DRAW1,160,0TO160,200:REM TRACCIA L´ASSE Y IN ROSSO
130 COLOR1,6:REM PASSA IN VERDE
140 DRAW1,0,199TO50,100TO90,50TO110,30TO150,20TO180,30
150 DRAW1,180,30TO220,10TO260,80TO320,0:REM TRACCIA LA CURVA DI INCREMENTO
160 CHAR1,5,23,"VENDITE PREVISTE FINO AL 1990"
170 CHAR1,0,21,"1970      1975      1980      1985      1990"
180 SLEEP10:GRAPHIC0,1:REM RITARDO E RITORNO IN MODO TESTO
```

GRAPHIC

Seleziona un modo grafico

1) GRAPHICmodo[,can][,d]
2) GRAPHICCLR

Quest'istruzione pone il C128 in uno di 6 modi grafici:

MODO	DESCRIZIONE
0	**testo a 40 colonne**
1	**bit map standard**
2	**bit map diviso**
3	**bit map multicolore**
4	**bit map multicolore diviso**
5	**testo a 80 colonne**

ESEMPI:

GRAPHIC1,1	Seleziona e cancella la bit map standard
GRAPHIC4,0,10	Seleziona la bit map multicolore divisa fino alla linea 10
GRAPHIC0	Seleziona il testo a 40 colonne
GRAPHIC5	Seleziona il testo a 80 colonne
GRAPHICCLR	Disalloca la bit map

PROGRAMMA DI ESEMPIO:

```
10 REM ESEMPIO DI MODI GRAFICI
20 COLOR0,1:COLOR4,1:COLOR1,7
30 GRAPHIC1,1:REM BIT MAP STANDARD
40 CIRCLE1,160,100,60,60
50 CIRCLE1,160,100,30,30
60 CHAR1,9,24,"MODO BIT MAP STANDARD"
70 SLEEP4
80 GRAPHIC0,1:REM TESTO STANDARD
90 COLOR1,6:REM PASSA IN VERDE
100 FORI=1TO25
110 PRINT"MODO TESTO STANDARD"
120 NEXT
130 SLEEP4
140 GRAPHIC2,1:REM SELEZIONA LO SCHERMO DIVISO
150 CIRCLE1,160,70,50,50
160 CHAR1,13,1,"SCHERMO DIVISO"
170 CHAR1,5,16,"MODO BIT MAP STANDARD IN ALTO"
180 FORI=1TO25
190 PRINT"MODO TESTO STANDARD IN BASSO"
200 NEXT
210 SLEEP3:REM RITARDO
220 SCNCLR:REM CANCELLA LO SCHERMO
230 GRAPHICCLR:REM DISALLOCA LA BIT MAP
```

GSHAPE

Recupera i dati da una variabile stringa e li traccia sullo schermo

GSHAPEstringa[,X,Y][,modo]

dove:

stringa	contiene la forma da disegnare
X,Y	coordinate dell'angolo superiore sinistro
modo	modo di tracciamento: 0=normale (default) 1=forma in reverse 2=OR tra forma e schermo 3=AND tra forma e schermo 4=XOR tra forma e schermo

PROGRAMMA DI ESEMPIO:

```
10 REM TRACCIA, SALVA E RITRACCIA IL SIMBOLO COMMODORE
20 COLOR0,1:COLOR4,1:COLOR1,7
30 GRAPHIC1,1:REM SELEZIONA LA BIT MAP
40 CIRCLE1,160,100,20,15:REM CERCHIO PIU´ ESTERNO
50 CIRCLE1,160,100,10,9:REM CERCHIO PIU´ INTERNO
60 BOX1,165,85,185,115:REM ISOLAMENTO DELL´AREA DA CANCELLARE
70 SSHAPEA$,166,85,185,115:REM SALVA L´AREA IN A$
80 GSHAPEA$,166,85,4:REM OR ESCLUSIVO CON L´AREA IN MODO DA SPEGNERE I PIXEL
90 DRAW0,165,94TO165,106:REM SPEGNE (TRACCIA NEL COLORE DI FONDO) I PIXEL IN C=
100 DRAW1,166,94TO166,99TO180,99TO185,94TO166,94:REM FLAG SUPERIORE
110 DRAW1,166,106TO166,101TO180,101TO185,106TO166,106:REM FLAG INFERIORE
120 PAINT1,160,110:REM RIEMPIE LA C
130 PAINT1,168,98:REM FLAG SUPERIORE
140 SLEEP5:REM RITARDO
150 SSHAPEB$,137,84,187,116:REM SALVA LA FORMA IN B$
160 DO
170 SCNCLR
180 Y=10
190 DO
200 X=10
210 DO
220 GSHAPEB$,X,Y:REM CARICA E TRACCIA LA FORMA
230 X=X+50:REM AGGIORNA LA X
240 LOOPWHILEX<280
250 Y=Y+40:REM AGGIORNA LA Y
260 LOOPWHILEY<160
270 SLEEP3
280 LOOP
```

LOCATE

Posiziona il cursore grafico (CP) sullo schermo

LOCATEX,Y

ESEMPI:

LOCATE160,100	Posiziona il CP al centro dello schermo senza tracciare
LOCATE+20,100	Muove il CP 20 pixel a destra all'altezza Y 100
LOCATE+30,+20	Muove il CP 30 pixel a destra e 20 in basso

PAINT

Riempie un'area di colore

PAINT[sorg col],X,Y[,modo]

dove:

sorg col	0=fondo 1=primo piano 2=multi 1 3=multi 2
X,Y	coordinate di partenza
modo	0=riempie un'area circondata dal colore selezionato 1=riempie un'area definita solo da pixel accesi

ESEMPI:

10 CIRCLE1,160,100,65	Traccia una circonferenza
20 PAINT1,160,100	Riempie il cerchio col colore di primo piano
10 BOX1,10,10,20,20	Traccia il contorno di un quadrato
20 PAINT1,15,15	Riempie il quadrato col colore di primo piano
30 PAINT1,+10,+10	Riempie un'area partendo da 10 pixel a destra e in basso dalla posizione precedente del CP

SCALE

Altera la scala dello schermo

SCALEn[,maxX,maxY]

dove:

n=1 (on) o 0 (off)
maxX = 320-32767 (default 1023)
maxY = 200-32767 (default 1023)

I valori di default non usando SCALE sono:

bit map standard	X (0-319), Y (0-199)
bit map multi	X (0-159), Y (0-199)

ESEMPI:

```
10 GRAPHIC1,1
20 SCALE1:CIRCLE1,180,100,100,100
```

Entra in modo grafico ponendo una scala di 1024x1024, poi traccia un cerchio

```
10 GRAPHIC1,3
20 SCALE1,1000,5000
30 CIRCLE1,180,100,100,100
```

Entra in grafica multicolore e pone la scala a 1000x5000, poi traccia il cerchio del programma precedente.

SSHAPE

Salva un'area in una variabile stringa

SSHAPE e **GSHAPE** si usano per salvare e ritracciare aree di bit map normale o multicolore in e da una variabile stringa. Per salvare un'area usate:

SSHAPEstringa,X1,Y1[,X2,Y2]

dove:

stringa	variabile contenitore
X1,Y1	coordinate dell'angolo superiore sinistro
X2,Y2	coordinate dell'angolo inferiore destro (default CP)

ESEMPI:

SSHAPEA$,10,10	Salva un'area rettangolare dalla coordinata 10,10 al CP nella variabile A$
SSHAPEB$,20,30,47,51	Salva un'area dalle coordinate 20,30 a 47,51
SSHAPED$,+10,+10	Salva un'area partendo da 10 pixel a destra e in basso dall'ultima posizione raggiunta dal CP

Inoltre guardate anche GSHAPE per un ulteriore esempio.

WIDTH

Pone la **larghezza dei punti tracciati**

WIDTHn

Questo comando stabilisce l'ampiezza dei punti tracciati coi comandi grafici. Dando n=1 si avrà un punto normale; con n=2 invece un punto doppio.

ESEMPI:

WIDTH1 Pone l'ampiezza singola
WIDTH2 Pone l'ampiezza doppia

6

LINGUAGGIO MACCHINA SUL COMMODORE 128

Questo capitolo vi introduce alla programmazione in linguaggio macchina del 6502. Leggete questa sezione se desiderate iniziare l'apprendimento del linguaggio macchina o se avete cominciato da poco. Questa sezione spiega i principi elementari che stanno dietro la programmazione in linguaggio macchina del vostro C128. Vi introduce anche al set di istruzioni di linguaggio macchina dell'8502 ed al modo d'uso di ognuna. Se siete già un esperto nella programmazione in linguaggio macchina, saltate questa sezione e continuate con la Tavola delle Istruzioni e degli Indirizzamenti dell'8502 in fondo al capitolo come materiale di riferimento. Il set di istruzioni dell'8502 è identico a quello del 6502.

COS'É IL LINGUAGGIO MACCHINA

Ogni computer ha il suo proprio linguaggio macchina. Il tipo di questo linguaggio dipende dal processore installato nel computer. Il vostro Commodore 128 interpreta il linguaggio macchina dell'8502, basato su quello del 6502, per eseguire le operazioni. Pensate al processore come al cervello del computer ed alle istruzioni come ai suoi pensieri.
Il linguaggio macchina è il livello più elementare di codice che il computer capisca. Il vero linguaggio macchina è composto da stringhe di zero e di uno.

Questi zero ed uno funzionano come degli interruttori per l'hardware e dicono ai circuiti dove applicare i voltaggi.
Il linguaggio macchina trattato in questo capitolo è il linguaggio simbolico del 6502 come appare nel Monitor di Linguaggio Macchina del C128. Questo non è il linguaggio simbolico completamente supportato come appare in un pacchetto Assembler, poichè etichette ed altre utilità ad alto livello non sono fornite.

Il linguaggio macchina è il linguaggio di più basso livello a cui potete istruire il vostro computer. Il BASIC è considerato un linguaggio ad alto livello. Sebbene il vostro C128 abbia il BASIC incorporato, il computer deve prima interpretarlo e tradurrlo al livello più basso di comprensione, prima di eseguire ogni istruzione BASIC.
Con ogni microistruzione, voi date al computer un dettaglio specifico da eseguire. Il computer non dà nulla per scontato in linguaggio macchina, al contrario del BASIC, dove vengono eseguite molte invisibili funzioni di macchina per ogni istruzione. Un'istruzione BASIC richiede parecchie istruzioni in linguaggio macchina per eseguire l'operazione medesima. In realtà, quando immettete un comando BASIC, state solo chiamando una subroutine in linguaggio macchina che esegue l'operazione.

PERCHÉ USARE IL LINGUAGGIO MACCHINA?

Se il linguaggio macchina è più intricato e complicato del BASIC, perchè usarlo? Certe applicazioni, come grafica e telecomunicazioni, richiedono il linguaggio macchina per la sua velocità. Poichè il computer non deve tradurre da un linguaggio ad alto livello, il programma va molte volte più veloce che in BASIC.
Programmi come quelli usati nei videogiochi non possono funzionare alla relativamente bassa velocità del BASIC, così sono scritti in linguaggio macchina. Altre applicazioni richiedono l'uso del linguaggio macchina semplicemente perchè le relative operazioni di programmazione sono svolte meglio che in BASIC. Ma alcune funzioni come le operazioni su stringhe sono svolte meglio in BASIC che in linguaggio macchina. In questi casi si possono usare assieme il BASIC ed il linguaggio macchina. Dentro il vostro computer c'è un programma perpetuamente in funzione chiamato sistema operativo. Il programma del sistema operativo controlla ogni funzione del vostro computer. Esso esegue le sue funzioni a velocità tali che non potete neanche immaginare.
Il programma del sistema operativo è scritto completamente in linguaggio macchina ed è memorizzato in una parte del computer chiamata ROM del Kernal (il Capitolo 14 [Vol. V] descrive come sfruttare i programmi in linguaggio macchina contenuti nel Kernal, e come usarne delle parti nei vostri programmi).
Sebbene all'inizio la programmazione in linguaggio macchina possa apparire più complicata e difficile del BASIC, ripensate a quando non conoscevate il BASIC o il vostro primo linguaggio di programmazione. Anche quello sembrava difficile, all'inizio. Se avete imparato il BASIC o un altro linguaggio di programmazione, potete imparare anche il linguaggio macchina. Nonostante sia una buona idea imparare un linguaggio ad alto livello prima di iniziare col linguaggio macchina, ciò non è assolutamente necessario.

COME APPARE IL LINGUAGGIO MACCHINA?

Il Capitolo 3 (Vol. I) descrive il linguaggio BASIC 7.0 del C128. Molte linee in BASIC iniziano con una parola chiave del BASIC, seguita da un operando. Le parole chiave del BASIC assomigliano all'inglese. Gli operandi sono variabili, o costanti, che fanno parte dell'espressione. Per esempio, A+B=2, è un'espressione dove A, B e 2 sono gli operandi. Le istruzioni in linguaggio macchina sono simili, sebbene abbiano un formato uniforme. Ecco il formato di un'istruzione simbolica dell'8502 come appare nel Monitor di Linguaggio Macchina del C128:

CAMPO DEL CODICE OPERATIVO E CAMPO DELL'OPERANDO

CAMPO DEL CODICE OPERATIVO (OP-CODE)

La prima parte di un'istruzione in linguaggio macchina viene chiamata codice operativo o op-code. L'op-code è paragonabile ad una parola BASIC, in quanto è la parte di istruzione che esegue l'operazione. Un op-code in linguaggio macchina è anche chiamato uno mnemonico. Tutti i mnemonici dell'assembler dell'8502 (6502) sono delle abbreviazioni di tre lettere riferentesi alla funzione eseguita. Per esempio, la prima e più comune istruzione che imparerete sarà LDA, che sta per LoaD the Accumulator (carica l'accumulatore). Questo capitolo spiegherà tutti i mnemonici.

CAMPO DELL'OPERANDO

La seconda parte di un'istruzione di linguaggio macchina è il campo dell'operando. Nel Monitor di Linguaggio Macchina del C128, l'operando è separato dall'op-code da almeno uno spazio e preceduto da un segno di dollaro ($), un più (+: decimale), una e romana (&: ottale), oppure un percento (%: binario) per indicare che l'operando è rispettivamente in forma esadecimale, decimale, ottale o binaria. Un'INDIRIZZO è il nome di una locazione di memoria contenente l'operando.
Il numero della locazione di memoria è il suo indirizzo, proprio come sono numerate le case della vostra strada. Gli indirizzi sono necessari nel vostro computer così da poter caricare, memorizzare ed inviare dati avanti e indietro dal microprocessore.
Quando usate il monitor interno di linguaggio macchina del C128, tutti i numeri ed indirizzi sono normalmente in formato esadecimale, ma potete specificarli anche come decimali, ottali o binari. L'indirizzo è il numero esadecimale di una specifica locazione di memoria. Quando usate un ASSEMBLATORE, gli indirizzi vengono presi come simbolici. Questo simbolismo vi permette di usare nomi di variabili invece di indirizzi fissi che specifichino la vera locazione di memoria. Voi all'inizio del programma in linguaggio macchina dichiarate l'equivalenza fra il simbolo e l'indirizzo vero e proprio oppure permettete all'assemblatore di assegnarlo subito.
Quando vi riferirete poi a quell'indirizzo nel corso del programma, potrete scrivere il simbolo equivalente piuttosto che l'indirizzo vero e proprio come nel Monitor di Linguaggio Macchina. Usare un assemblatore ed indirizzi simbolici rende la programmazione più semplice che usando il monitor di linguaggio macchina e gli indirizzi veri. Più avanti in questo capitolo apprenderete sui 13 modi di indirizzamento presenti.
Come sapete, la seconda parte di un'istruzione in linguaggio macchina è l'OPERANDO. Un operando può essere una costante; non deve necessariamente riferirsi ad un indirizzo. Quando al posto di un indirizzo appare una costante viene effettuata un'operazione su un dato piuttosto che su una locazione di memoria.
Un segno # davanti all'operando specifica un indirizzamento immediato, di cui saprete altro più avanti nel capitolo. Il segno # è usato solo come aiuto simbolico per il programmatore. Questo segno dice al computer di effettuare un'opera-

zione su una costante e non su un indirizzo. Nel caso del Monitor di Linguaggio Macchina non sono permessi nomi di variabili. Per rappresentare delle variabili nel monitor, dovete riferirvi alla locazione di memoria contenente il dato.

ESEMPI DI ISTRUZIONI IN LINGUAGGIO MACCHINA

```
LDA $0100        ; Indirizzamento assoluto
LDA $10          ; Indirizzamento in pagina zero
LDA ($FA),Y      ; Indirizzamento indiretto indicizzato
LDA $2000,X      ; Indirizzamento assoluto indicizzato
LDA #$10         ; Indirizzamento immediato (costanti)
```

SOMIGLIANZE E DIFFERENZE FRA UN ASSEMBLATORE ED UN MONITOR DI LINGUAGGIO MACCHINA

Entrambi supportano gli op-code simbolici. Gli assemblatori tipicamente permettono gli operandi simbolici, mentre invece il monitor fa letteralmente riferimento ad indirizzi ed operandi reali.
Un assemblatore ha generalmente due tipi di file: codice sorgente e codice oggetto. Il codice sorgente viene scritto da voi con etichette e commenti.
Il sorgente non è eseguibile direttamente. Deve essere assemblato (con un processo intermedio) nel codice oggetto, che è direttamente eseguibile.
L'indirizzo di partenza nel monitor è determinato dall'indirizzo da cui si inizia a scrivere il programma. Non potete inserire commenti. Il programma risultante, una volta immesso, può essere eseguito immediatamente come file binario e non richiede passi intermedi di assemblaggio.

I REGISTRI DEL MICROPROCESSORE 8502

Avete imparato che un indirizzo è un riferimento ad una specifica locazione di memoria fra quelle dei due banchi di RAM del C128. Nel microprocessore stesso esistono però delle aree separate ed indipendenti dalla RAM usate per scopi speciali di lavoro e memorizzazione e chiamate registri. Questi registri sono dove i valori vengono manipolati. La manipolazione dei registri del microprocessore e la loro comunicazione con la memoria del computer (RAM e ROM) sono le funzioni svolte dal linguaggio macchina e dal sistema operativo del vostro computer.

La Figura 6-1 mostra un diagramma a blocchi del microprocessore 8502. Come visto nella figura, i registri dell'8502 sono:

Accumulatore
Registro indice X
Registro indice Y
Registro di stato
Contatore di programma
Puntatore allo stack

Di seguito vi sono le descrizioni di questi registri.

Figura 6-1. Diagramma a blocchi dell'8502

L'ACCUMULATORE

L'accumulatore è uno dei più importanti registri all'interno dell'8502. Come implicito nel nome, esso accumula i risultati di specifiche operazioni. Pensate all'accumulatore come al passaggio verso il microprocessore. Tutte le informazioni che entrano nel vostro computer devono prima passare attraverso l'accumulatore (o i registri X e Y).
Per esempio, se volete memorizzare un valore in una locazione della RAM, dovete prima caricarlo nell'accumulatore (o nel registro X o Y) e poi immagazzinarlo nella locazione RAM specificata. Non potete scrivere un valore direttamente nella RAM senza piazzarlo prima nell'accumulatore o nei registri indice (i registri indice sono descritti nella sezione seguente).

Tutte le operazioni matematiche sono eseguite dentro l'unità aritmetico-logica (ALU) ed il risultato messo nell'accumulatore. Viene considerata un'area temporanea di lavoro matematico. Per esempio, volete sommare due numeri, 2+3. Prima caricate l'accumulatore con 2; poi aggiungete 3 col mnemonico ADC. Ora, volete fare un'altra operazione. Dovete depositare il risultato dall'accumulatore in una locazione RAM prima di eseguire la prossima operazione. Se non lo fate perderete il primo risultato.
L'accumulatore è così importante che possiede un modo di indirizzamento tutto suo. Tutte le istruzioni che usano questo modo si riferiscono specificamente all'accumulatore. Le seguenti tre istruzioni campione fanno riferimento solamente all'accumulatore nel suo proprio modo di indirizzamento:

LDA - Carica l'accumulatore dalla memoria
STA - Salva l'accumulatore in memoria
ADC - Aggiunge il contenuto della memoria all'accumulatore

I dettagli su quel che riguarda i modi di indirizzamento dell'accumulatore sono dati più avanti nel capitolo.

I REGISTRI X E Y

I registri più usati dopo l'accumulatore sono i registri indice X e Y. Questi registri indice sono usati principalmente entro un'instruzione per modificare un indirizzo aggiungendo un indice. Possono essere anche usati come locazioni di salvataggio temporaneo o per caricare e memorizzare valori nella RAM.
Nella modifica di un indirizzo, i contenuti del registro indice vengono aggiunti all'indirizzo originale, chiamato indirizzo di base, per calcolare il nuovo indirizzo relativo a quello di base. L'indirizzo risultante produce quello effettivo - cioè, la locazione dove il dato verrà posto o da dove verrà prelevato. L'indirizzo effettivo sarà poi usato dalle istruzioni. Per esempio, volete porre il valore 0 nelle locazioni da 1024 a 1034. In BASIC ecco come dovete fare:

```
10 FORI=1024TO1034
20 POKEI,0
30 NEXT
```

Ecco come dovete farlo in un linguaggio macchina simbolico usando il registro X o Y. NOTA: non preoccupatevi se non capite tutte le istruzioni seguenti. Saranno trattate a fondo nella sezione TIPI DI ISTRUZIONI, più avanti nel capitolo.

```
        LDA #$00     Carica 0 nell'accumulatore
        TAX          Trasferisce il contenuto dell'accumulatore nel
                     registro X
START   STA $0400,X  Pone il contenuto dell'accumulatore in 0400
                     + X
        INX          Incrementa il registro X
        CPX #$0B     Compara X con 11
        BNE START    Se X non è uguale salta a START
        BRK          Ferma il programma
```

* = Nel monitor di linguaggio macchina l'etichetta START non è permessa, così l'etichetta apparirà come un indirizzo reale (p.e.: $183B).

L'esempio BASIC sopra pone uno 0 nelle locazioni da 1024 a 1034. La linea 10 imposta un ciclo dalle locazioni 1024 a 1034. La linea 20 pone il valore 0 nelle locazioni specificate da I. La prima volta I equivale a 1024, la seconda a 1025 e così via. La linea 30 incrementa la variabile I di 1 ogni volta.
L'esempio precedente in linguaggio macchina compie lo stesso lavoro dell'esempio BASIC. LDA #$00 carica uno 0 nell'accumulatore. TAX trasferisce il contenuto di A in X. Le istruzioni seguenti formano un ciclo:

```
START   STA $0400,X
        INX
        CPX #0B
        BNE START
```

Ecco cosa avviene nel ciclo. STA $0400,X pone uno 0 nella locazione (esa) $0400 la prima volta. $0400 equivale a 1024. INX aggiunge 1 a X ogni volta che viene eseguito. CPX #$0B confronta i contenuti di X con la costante $0B (11). Se X non contiene 11 allora il programma salta a START STA $0400,X e il ciclo viene ripetuto.
La seconda volta 0 viene posto in $0400 + X, cioè $0401 (1025), e X viene incrementato nuovamente. Il ciclo continua finchè X non raggiunge 11.
L'indirizzo effettivo del primo ciclo è $0400, cioè 1024 in decimale. La seconda volta l'indirizzo è $0400 + 1, e così via. Ora avete visto come i registri indice possono modificare l'indirizzo in un'istruzione in linguaggio macchina.

IL REGISTRO DI STATO

Il registro di stato del processore indica lo stato di certe condizioni dell'8502. Viene controllato da 7 istruzioni ed indica le condizioni tramite dei flag. Essendo composto da un byte, ogni bit indica una condizione, eccetto il bit 5.
Le istruzioni di diramazione controllano 4 bit per determinare se una certa condizione è in atto. La condizione per saltare fa riferimento allo stato dei bit: se il bit è 1, allora la condizione è vera e viene effettuata la diramazione. Se, al contrario, la condizione è falsa, il computer non salterà e continuerà dall'istruzione seguente quella di test.
La figura 6-2 mostra lo schema del registro di stato dell'8502 e le condizioni dei flag.

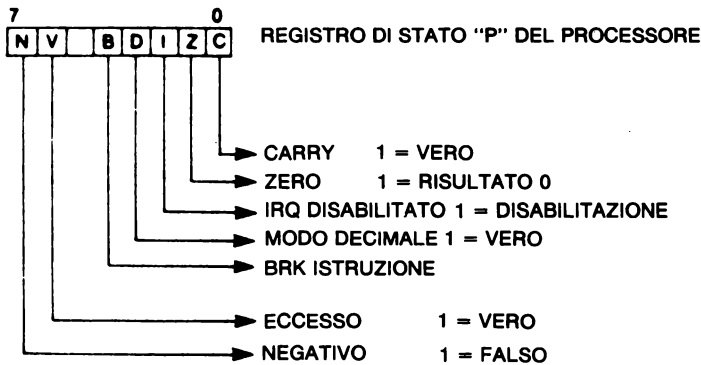



Figura 6-2. Registro di stato dell'8502

Il bit carry (0) vale 1 se un'addizione ha dato un risultato maggiore di 255, cioè a 9 bit: 8 bit stanno in A, il non va nel carry. Questo bit vale 1 anche in altre condizioni. Per metterlo a 1 usate SEC (SEt Carry), per metterlo a 0 CLC (CLear Carry).
Il bit zero (1) va a 1 se un'operazione ha dato un risultato zero. BEQ testa la condizione e salta se 0. BNE salta invece se non 0. Se il bit zero vale 0 allora il comando BEQ non viene eseguito e si passa all'istruzione seguente.
Il bit IRQ (2) se 1 disabilita le richieste di interruzione: SEI lo pone a 1, CLI lo pone a 0 e permette le interruzioni. Se date l'istruzione SED (SEt Decimal) il microprocessore passa in modo decimale. Per tornare in modo binario azzerate il bit 3 con l'istruzione CLD (CLear Decimal).
Il flag di BRK (bit 4) ha un funzionamento simile al bit di IRQ (2) e passa a 1 con un'istruzione BRK. Tale istruzione ha lo stesso effetto di un'interruzione e causa il deposito sullo stack del Contatore di Programma e del registro di stato. Il sistema operativo riconoscerà poi se l'interruzione è stata generata via IRQ o da BRK ed eseguirà la routine appropriata.

Nel caso il flag di BRK fosse azzerato, il processore tratterebbe il tutto come una normale interruzione. Al contrario però di un'interruzione, BRK salva il Contatore di Programma aggiungendo due all'indirizzo. Il processore si aspetta poi di trovare la prossima istruzione due byte più avanti di BRK, che invece ne occupa uno solo. Potreste perciò aver bisogno di correggere l'indirizzo perchè potrebbe anche non corrispondere alla vera istruzione da eseguire.
Il bit di eccesso (6) è 1 se un'operazione ha cambiato lo stato del bit 7 dell'operando (bit di segno). Usate l'istruzione CLV (CLear oVerflow) per azzerarlo. Per testarlo ci sono invece BVS (salta se V a 1) e BVC (salta se V a 0). Potete usare l'istruzione BIT per metterlo a 1 di proposito.
Il flag di negativo (7) vale 1 se il risultato di un'operazione è minore di 0. Potete testarne lo stato con BMI (salta se 1) e BPL (salta se 0).
Il registro di stato indica 7 importanti condizioni del processore durante l'esecuzione del vostro programma, che può testarne alcune e agire di conseguenza. Vi dà così modo di controllare certe funzioni della macchina dipendenti dal valore dei flag.

IL CONTATORE DI PROGRAMMA

Finora tutti i registri dell'8502 erano a 8 bit, o un byte. Il contatore di programma è doppio in larghezza (16 bit) dell'accumulatore, dei registri X o Y e del registro di stato. Il contatore di programma è a 16 bit perchè contiene l'indirizzo corrente della prossima istruzione da eseguire. Gli indirizzi usati in un processore del tipo dell'8502 sono tutti ampi 16 bit. Devono essere così in modo da poter indirizzare tutte le locazioni entro un banco di 64 Kbyte di RAM.
Il contatore di programma mantiene l'indirizzo della prossima istruzione da eseguire. Da esso vengono prelevati sequenzialmente (di solito) gli indirizzi delle istruzioni e messi sul bus degli indirizzi a 16 bit. Il processore ottiene poi sul bus dei dati le istruzioni o i dati contenuti all'indirizzo specificato. Infine essi vengono decodificati ed eseguiti.

IL PUNTATORE ALLO STACK

Nella RAM del C128 c'è un'area di lavoro temporanea chiamata stack (catasta). Va dalla locazione 256 alla 511 ($0100-$01FF). Quest'area della RAM del computer è conosciuta come pagina 1. La paginazione sarà spiegata nella prossima sezione.
Lo stack è usato per tre scopi nel vostro computer: memorizzazione temporanea (salvataggio), controllo delle subroutine, e interruzioni. Lo stack è una struttura LIFO (Last In First Out, ultimo dentro primo fuori), il che significa che l'ultimo dato immesso è il primo ad essere prelevato. Quando ponete un dato sullo stack, si parla di push (spingi), mentre quando lo prelevate di pull o pop (tira).
Pensate a questa struttura come una catasta di vassoi da pranzo in una tavola calda. Il primo che viene usato è quello tirato su dalla cima. L'ultimo usato è quello in fondo, e viene prelevato solo dopo tutti gli altri.
Il puntatore allo stack indica l'indirizzo del primo byte libero sulla catasta. Quando prelevate un dato, il puntatore viene prima incrementato e poi il byte viene tirato su; così il puntatore indica sempre il primo byte libero. L'in-

verso avviene ponendo un byte: il byte viene salvato, poi il puntatore è decrementato. Poichè il puntatore parte sempre dal valore 255, tutti i 256 byte dello stack vengono utilizzati. Quando chiamate una subroutine oppure avviene un'interruzione, l'indirizzo al momento della chiamata viene salvato sullo stack, in modo che, una volta eseguita la subroutine o l'interruzione, l'esecuzione del programma riprende da dove era stata interrotta.

INDIRIZZAMENTO A 16 BIT: IL CONCETTO DI PAGINAZIONE

Il Commodore 128 contiene 128 Kbyte di Memoria ad Accesso Casuale (RAM). Ciò significa che voi avete due banchi di 65536 (64 Kbyte) locazioni di memoria RAM (meno 2 per le locazioni 0 e 1 sempre presenti in ciascun banco). Poichè l'8502 è un processore a 8 bit, necessita di due byte per rappresentare ogni numero fra 0 e 65535. Un byte può rappresentare solo numeri fra 0 e 255. Ci vuole dunque un modo per rappresentare tutte le locazioni di memoria.
Ecco come il vostro computer rappresenta il più grande numero in un byte. Esso viene memorizzato come numero binario, che voi solitamente rappresentate in formato esadecimale. La Figura 6-3 mostra le relazioni fra numeri binari, esadecimali e decimali.

BINARIO	ESADECIMALE	DECIMALE
11111111	$FF	255

Figura 6-3. Comparazione fra sistemi di numerazione

Un byte contiene 8 cifre binarie (bit). Ogni byte può avere valore 0 o 1. Il più grande numero che il vostro computer può rappresentare in 8 bit è 11111111, equivalente a 255 decimale. Questo significa che tutti gli 8 bit sono a 1. Nella conversione da binario a decimale, ogni bit a 1 equivale a 2 elevato la posizione del bit. Questa posizione è etichettata da 7 a 0 da sinistra a destra. La Figura 6-4 fornisce una rappresentazione visiva della conversione da binario a decimale.

	2↑7	2↑6	2↑5	2↑4	2↑3	2↑2	2↑1	2↑0
=	1	1	1	1	1	1	1	1
=	128 +	64 +	32 +	16 +	8 +	4 +	2 +	1 = 255

Figura 6-4. Conversione binario/decimale

Ogni riga rappresenta il valore di 2 elevato alla posizione del bit e moltiplicato per il valore: se un bit è 1 allora si somma il valore, altrimenti il risultato è nullo (qualunque valore per 0 dà 0). La figura 6-5 mostra un altro esempio di conversione binario/decimale.

	2↑7	2↑6	2↑5	2↑4	2↑3	2↑2	2↑1	2↑0
=	1	1	0	0	1	0	1	0
=	128 +	64 +	0 +	0 +	8 +	0 +	2 +	0 = 202

Figura 6-5. Conversione binario/decimale

Ricordate che solo i bit 1 danno un valore diverso da 0.
Ora che sapete convertire un byte da binario a decimale, starete probabilmente fantasticando su cosa possa avere a che fare con l'indirizzamento a 16 bit. Abbiamo detto prima che il contatore di programma - il registro contenente l'indirizzo dell'istruzione da eseguire - è ampio 16 bit. Questo significa che usa 2 byte adiacenti per calcolare l'indirizzo.
Avete imparato proprio adesso cos'è un byte basso, cioè la metà più bassa dei 16 bit usati per un indirizzo. La metà più alta è chiamata byte alto. Il byte alto rappresenta la metà alta di un indirizzo esattamente come la metà bassa, eccetto che le posizioni dei bit sono numerate da 8 a 15 da destra a sinistra. Quando sommate le potenze di 2 per queste posizioni di bit al valore del byte basso, arrivate ad indirizzi fino a 65535. Grazie a ciò il vostro computer può rappresentare ogni numero fra 0 e 65535, e gli indirizzi per ogni locazione di memoria di ogni banco da 64 Kbyte. La Figura 6-6 illustra un indirizzo a 16 bit in decimale:

Byte alto		2↑15	2↑14	2↑13	2↑12	2↑11	2↑10	2↑9	2↑8
	=	1	1	1	1	1	1	1	1
	=	32768	16384	8192	4096	2048	1024	512	256 = 65280
		+	+	+	+	+	+	+	
Byte basso		2↑7	2↑6	2↑5	2↑4	2↑3	2↑2	2↑1	2↑0
	=	1	1	1	1	1	1	1	1
	=	128 +	64 +	32 +	16 +	8 +	4 +	2 +	
							255	65280+255=65535	

Figura 6-6. Esempio di indirizzo a 16 bit in decimale

Potete vedere come il più alto numero rappresentabile dal byte alto di un indirizzo a 16 bit è 65280. E sapete che il maggior numero rappresentabile dal byte basso è 255. Aggiungete il byte alto al byte basso (65280+255) ed arriverete a 65535, il più alto indirizzo in un banco di 64 Kbyte.
Quando il microprocessore calcola l'indirizzo della prossima istruzione, guarda il byte alto del contatore di programma. Provate a pensarci come ad un normale byte. In questo caso, le posizioni dei bit saranno numerate da 7 a 0, proprio come il byte basso, e si potrà rappresentare un numero fino a 255.
Il valore del byte alto determina a quale blocco di 256 byte si accede. Questi blocchi sono chiamati pagine. Il byte alto determina nell'indirizzo l'inizio della pagina, così viene incrementato ogni 256 byte. Questo byte determina dunque a quale pagina di 256 byte si sta accedendo. Poichè un byte può contenere al

massimo 255, se voi moltiplicate questo valore per il numero di byte di ogni pagina, 256, ottenete che l'ultima pagina inizia a 65280, lo stesso numero del byte alto in Figura 6-6. 65280 è dunque il massimo indirizzo di inizio pagina.
Cosa succede se volete indirizzare una locazione di memoria che non cade all'inizio di una pagina? Ecco dove interviene il byte basso.
Il byte alto del contatore di programma rappresenta l'inizio delle pagine, mentre tutti gli indirizzi all'interno di ogni pagina sono rappresentati dal byte basso. Per esempio, l'indirizzo 65380 è rappresentato dal byte alto 255, che arriva fino a 65280, più il byte basso 100, che così sommato fa 65380.
Quando osservate la mappa di memoria del vostro C128, vedrete dei riferimenti a dei puntatori o vettori in forma di byte basso/alto a certe routine in linguaggio macchina entro il sistema operativo o ad importanti locazioni di sistema, come l'inizio del BASIC.
Potete trovare i contenuti di questi indirizzi e dove risiedono le routine nella memoria del vostro C128 usando il comando PEEK in BASIC, o il comando Memory del Monitor di Linguaggio Macchina. Per trovare l'indirizzo effettivo col BASIC, cercate nella mappa di memoria il riferimento ad una specifica routine o funzione del sistema, chiamato spesso vettore. Fate una PEEK del byte alto, il numero di pagina della routine, e moltiplicatelo per 256. Poi fate una PEEK del byte basso ed aggiungetelo al risultato del byte alto per trovare l'indirizzo decimale effettivo.
Tenete a mente che tutti i calcoli degli indirizzi sono effettuati in binario. Sono convertiti in decimale per facilità di comprensione. Nei vostri programmi in linguaggio macchina vi riferirete alla memoria sempre in notazione esadecimale, spiegata nella prossima sezione.

LA NOTAZIONE ESADECIMALE

Il vostro microprocessore 8502 capisce soltanto le cifre binarie 0 e 1. Sebbene il linguaggio macchina richieda usualmente la notazione esadecimale ed il BASIC quella decimale, tutti quei numeri sono tradotti e processati come numeri binari. Il vostro computer usa tre differenti sistemi di numerazione, binario (base 2), esadecimale (base 16) e decimale (base 10). Il monitor di linguaggio macchina usa anche la notazione ottale (base 8). Una base numerica è chiamata anche radice; quindi, il C128 usa 4 radici, ma il processore capisce solo il binario.
Il BASIC capisce i numeri decimali perchè sono facili da usare per le persone normali. Nonostante il BASIC non giri veloce quanto il linguaggio macchina, la facilità d'uso compensa la perdita di velocità.
Il linguaggio macchina usa la notazione esadecimale perchè è più vicina al sistema di numerazione binaria e più facile da tradurre rispetto al sistema decimale. La rappresentazione esadecimale è usata solitamente anche dai programmatori in linguaggio macchina perchè è più facile pensare ad un gruppo di 8 cifre binarie (un byte) che ad ogni singola cifra. Come trovate più facile rappresentare questo valore:

3A (esadecimale) o 00111010 (binario)?

Una volta che i valori sono tradotti dal linguaggio ad alto livello in una forma che il microprocessore possa comprendere (cifre binarie o bit), essi sono interpretati dalla circuiteria interna come degli interruttori. Questi interruttori determinano se il circuito integrato (I.C.) dovrà trasmettere un impulso elettronico per eseguire una funzione specifica, come indirizzare una locazione di memoria. Se il bit vale 1, l'interruttore è chiuso, cosa che invia attraverso l'I.C. un voltaggio fra 3 e 5 volt. Se viceversa la cifra binaria è 0 non viene trasmesso nessun impulso. Sebbene questa sia una spiegazione molto semplificata, potete farvi un'idea di come il microcomputer possa tradurre, decodificare ed eseguire le istruzioni che gli date. L'hardware ed il software si incontrano qui, al livello della macchina.

CAPIRE LA NOTAZIONE ESADECIMALE (HEX)

Il segreto per una facile comprensione dell'esadecimale (base 16) consiste nel dimenticare il decimale (base 10). Le cifre esadecimali sono numerate da 0 a 9 e continuano da A a F, dove F equivale a 15 decimale. Per convenzione, i numeri esadecimali si scrivono con un segno di dollaro ($) davanti in modo da distinguerli dai valori decimali. La Figura 6-7 fornisce una tabella di equivalenza fra cifre esadecimali, decimali e binarie:

ESADECIMALE	DECIMALE	BINARIO
$0	0	0000
$1	1	0001
$2	2	0010
$3	3	0011
$4	4	0100
$5	5	0101
$6	6	0110
$7	7	0111
$8	8	1000
$9	9	1001
$A	10	1010
$B	11	1011
$C	12	1100
$D	13	1101
$E	14	1110
$F	15	1111

Figura 6-7. Conversione esadecimale/decimale/binario

Ogni cifra hex (esa) rappresenta 4 bit. Il maggior numero rappresentabile con 4 bit è 15 decimale. In linguaggio macchina di solito si rappresentano operandi ed indirizzi come 2 o 3 cifre esa. Poichè in un indirizzo esadecimale a 4 cifre ogni cifra impiega 4 bit, queste 4 cifre rappresentano i 16 bit di indirizzamento. Dapprima vi troverete a convertire indirizzi ed operandi decimali in esadecimale. Poi vorrete fare la conversione opposta. Guardate le funzioni HEX e DEC per delle facili e veloci conversioni esadecimali. Nel monitor di linguaggio macchina, usate il segno più (+) per rappresentare i numeri decimali. Per ora usate le conversioni, ma col tempo dovreste trovarvi a pensare in esadecimale invece di convertire sempre.

MODI DI INDIRIZZAMENTO NEL COMMODORE 128

L'indirizzamento è il processo mediante il quale il microprocessore fa riferimento alla memoria. Il microprocessore 8502 ha molti modi di indirizzare le locazioni interne di memoria. I differenti modi di indirizzamento richiedono sia 1, 2 od anche 3 byte per ogni istruzione. Ogni istruzione ha un formato ed un op-code differenti. Per esempio, LDA (carica l'accumulatore) ha 8 formati, ognuno con un op-code diverso per specificare i vari modi d'indirizzamento. Guardate nella sezione con la Tavola delle Istruzioni e dell'Indirizzamento dell'8502 per vedere i differenti formati di tutte le istruzioni.

INDIRIZZAMENTO DELL'ACCUMULATORE

L'indirizzamento dell'accumulatore implica che il codice operativo specificato agisce sull'accumulatore. Il campo dell'operando viene omesso dacchè l'istruzione opera solo sull'accumulatore; così l'op-code è limitato ad un solo byte. Ecco alcuni esempi di istruzioni con indirizzamento dell'accumulatore:

ISTRUZIONE	CODICE HEX	SIGNIFICATO
ASL	$0A	Scorrimento logico a sinistra
LSR	$4A	Scorrimento logico a destra
ROR	$6A	Rotazione a destra

INDIRIZZAMENTO IMMEDIATO

L'indirizzamento immediato specifica che l'operando è una costante piuttosto che il contenuto di un certo indirizzo. L'operando è il dato stesso, non il puntatore al dato. A livello di macchina, il microprocessore in realtà distingue una costante ed un indirizzo da due diversi op-code, mentre il programmatore deve usare il segno # per identificare la costante dall'indirizzo. Le istruzioni ad indirizzamento immediato richiedono 2 byte. Ecco alcuni esempi di istruzioni ad indirizzamento immediato:

ISTRUZIONE	CODICE HEX	SIGNIFICATO
LDA #$0F	$A9	Carica 15 nell'accumulatore
CMP #$FF	$C9	Confronta l'accumulatore con 255
SBC #$E0	$E9	Sottrae 224 dall'accumulatore

INDIRIZZAMENTO ASSOLUTO

L'indirizzamento assoluto vi permette di accedere ad ogni locazione di memoria fra quelle contenute in ogni banco di 64 Kbyte. Questo indirizzamento richiede 3 byte; il primo per l'op-code, il secondo per il byte basso ed il terzo per quello alto. Ecco alcuni esempi di istruzioni ad indirizzamento assoluto:

ISTRUZIONE	CODICE HEX	SIGNIFICATO
INC $4FFC	**$EE**	**Aggiunge 1 al contenuto di $4FFC**
LDX $200C	**$AE**	**Carica in X il contenuto di $200C**
JSR $FFC3	**$20**	**Chiama la routine che inizia a $FFC3**

INDIRIZZAMENTO DI PAGINA ZERO

L'indirizzamento di pagina zero richiede 2 byte; il primo per l'op-code ed il secondo per l'indirizzo in pagina zero. Poichè la pagina zero varia solo da 0 a 255, il processore ha bisogno solo del byte basso dell'indirizzo. Il byte alto viene assunto uguale a 0 e quindi non viene specificato. Potete usare lo stesso l'indirizzamento assoluto per indirizzare la pagina zero, ma richiedendo quest'ultimo 3 byte non è veloce come quello in pagina zero. Ecco alcuni esempi:

ISTRUZIONE	CODICE HEX	SIGNIFICATO
LDA $FF	**$A5**	**Carica nell'accumulatore i contenuti di 255**
ORA $E4	**$05**	**Esegue un OR fra l'accumulatore e il contenuto di 228**
ROR $0F	**$66**	**Rotazione a destra del contenuto di 15**

INDIRIZZAMENTO IMPLICITO

Nel modo d'indirizzamento implicito non viene specificato nessun operando poichè l'op-code indica da solo l'azione da fare. Poichè non esistono nè operandi od indirizzi il codice è limitato ad 1 byte. Ecco qualche esempio:

ISTRUZIONE	CODICE HEX	SIGNIFICATO
DEX	**$CA**	**Sottrae 1 a X**
INY	**$C8**	**Aggiunge 1 a Y**
RTS	**$60**	**Ritorna da una subroutine**

INDIRIZZAMENTO RELATIVO

L'indirizzamento relativo viene usato solo con le istruzioni di diramazione. Queste istruzioni (BEQ, BNE, BCC ecc.) vi permettono di alterare la via di esecuzione di un programma in relazione a certe condizioni. Le istruzioni di diramazione sono simili alle IF...THEN del BASIC poichè entrambe eseguono un determinato insieme di istruzioni.
L'operando nell'istruzione di diramazione determina la destinazione del salto condizionato. Per esempio, l'istruzione BEQ sta per Branch on result EQual to zero (salta se il risultato è zero). Se il flag zero vale 1, allora l'istruzione aggiunge l'operando al contatore di programma e continua l'esecuzione dal nuovo indirizzo calcolato. La figura 6-8 fornisce un esempio in linguaggio assembly simbolico.

```
        LDA #01      .01800 A9 01 LDA #01
        STA TEMP     .01802 85 FA STA $FA
START   DEC TEMP     .01804 C6 FA DEC $FA
        BEQ START    .01806 F0 FC BEQ $1804
        LDX #01      .01808 A2 01 LDX #01
        STA COUNT    .0180A 85 FB STA $FB
```

NOTA: il monitor di linguaggio macchina non permette etichette simboliche come START e COUNT.

Figura 6-8. Indirizzamento relativo

La Figura 6-8 elenca il codice (A) a sinistra così come appare nell'assembly simbolico. Il codice (B) in mezzo è il vero codice macchina così come appare nel monitor. Il codice (C) a destra è il liguaggio simbolico così come lo accetta il monitor.
In questo segmento di programma, la prima istruzione carica 1 nell'accumulatore. STA è l'op-code che memorizza il contenuto di A in TEMP. La terza istruzione, DEC, sottrae 1 a TEMP. Nella quarta istruzione START è un'etichetta che segna l'inizio del ciclo condizionale. L'istruzione di salto (BEQ) controlla se il valore contenuto in TEMP è 0 in seguito all'istruzione DEC. L'istruzione BEQ segna la fine del ciclo.
La prima volta che viene eseguito il ciclo, TEMP è zero, per cui viene eseguito il salto a START.
La seconda volta TEMP è minore di zero; quindi il flag zero nel registro di stato varrà 0 e il programma non eseguirà il salto, continuando dalla prossima istruzione (LDX #01).

A causa del modo in cui è scritto il programma, il salto può essere effettuato una sola volta.
Nell'indirizzamento relativo, il primo byte dell'istruzione è l'op-code ed il secondo l'operando, che rappresenta uno spostamento di n locazioni di memoria. La locazione a cui saltare non viene interpretata come un indirizzo assoluto ma come uno spostamento relativo all'indirizzo dell'istruzione di salto.
Lo spostamento va da -128 a 127. Se si raggiunge la condizione di salto, allora lo spostamento viene aggiunto al contatore di programma e l'esecuzione riprende dal nuovo indirizzo.
Nell'esempio di Figura 6-8, notate come l'istruzione di salto sia dopo l'etichetta START. L'operando è interpretato come uno spostamento di 3 byte indietro in memoria poichè si devono saltare l'op-code di BEQ (1 byte) e l'istruzione DEC (2 byte). L'8502 può solo saltare 127 byte avanti e 128 indietro.
Se entrate nel monitor di linguaggio macchina e disassemblate il codice, potrete vedere come il computer rappresenta un operando di un'istruzione di salto, come nella colonna (B) della Figura 6-8. Il codice simbolico nel campo operando della parte (C) rappresenta un indirizzo assoluto ma il codice esadecimale assemblato della parte (B) usa per questo indirizzo un solo byte, uno spostamento positivo o negativo dall'indirizzo dell'istruzione di salto. Il più grande numero di spostamenti in avanti è 127. Uno spostamento indietro è invece rappresentato da un numero maggiore di 128. Quando siete nel monitor di linguaggio macchina, sottraete lo spostamento da 255 (FF) per trovare il valore reale dello spostamento negativo. In questo caso FF-3 uguale FC, che è proprio l'operando nell'istruzione di salto in colonna (B) della figura 6-8.
Ecco alcuni esempi di istruzioni di salto mediante indirizzamento relativo:

ISTRUZIONE	CODICE HEX	SIGNIFICATO
BEQ	**$F0**	**Salta se uguale a zero**
BNE	**$D0**	**Salta se non uguale a zero**
BCC	**$90**	**Salta se il carry è 0**

MODI DI INDIRIZZAMENTO INDICIZZATO

Il Commodore 128 ha 2 registri a scopi speciali: i registri indice X e Y. Nei modi d'indirizzamento indicizzato, i registri indice modificano un indirizzo aggiungendo il loro contenuto all'indirizzo di base per arrivare a quello effettivo. Per esempio, ecco un segmento di programma che illustra l'importanza della modifica degli indirizzi, usando i registri X e Y:

```
        LDA #$0F
        LDX #$00
LOOP    STA $2000,X
        INX
        BNE LOOP
```

La prima istruzione carica 15 nell'accumulatore. La seconda 0 nel registro X. La terza pone il contenuto dell'accumulatore all'indirizzo $2000 + il contenuto del registro X. La prima volta X vale 0, quindi l'indirizzo effettivo è $2000 + 0. L'istruzione INX incrementa poi X, e BNE salta a LOOP finchè X non è zero. Alla seconda esecuzione del ciclo, X vale 1, perciò A è posto a $2000 + 1, cioè $2001. La terza volta a $2002, e così via.

Il ciclo continua e pone 15 in tante locazioni consecutive finchè X non diventa 0. In altre parole, il ciclo viene eseguito 256 volte, da 0 a 255; poi, 255 + 1 = 0, e il ciclo finisce. Aggiungendo 1 a 255 il registro diventa 0, poichè si genera un riporto verso un nono bit che non può essere contenuto. È un comportamento simile a quando il contachilometri della vostra macchina arriva a 99999 chilometri. Il prossimo chilometro che fate esso passa da 99999 a 00000.
Questo esempio mostra solo un modo di modificare gli indirizzi tramite i registri indice. Il C128 ha 4 modi di indirizzamento indicizzato: 1) indicizzato assoluto (visto nell'esempio sopra), 2) indicizzato in pagina zero, 3) indicizzato indiretto e 4) indiretto indicizzato.

INDIRIZZAMENTO INDICIZZATO IN PAGINA ZERO

Questo tipo di indirizzamento è simile a quello in pagina zero, eccetto che i registri X e Y sono usati per modificare l'indirizzo. Anche questo tipo di indirizzamento richiede solo 2 byte. L'indirizzo effettivo è calcolato aggiungendo il contenuto di un registro indice al byte basso dell'indirizzo. Questo tipo di indirizzamento è ovviamente più veloce di quello indicizzato assoluto.
Ecco alcuni esempi di istruzioni indicizzate in pagina zero:

ISTRUZIONE	CODICE HEX	SIGNIFICATO
INCoper.,X	**$F6**	**Aggiunge 1 al contenuto dell'operando. L'operando è l'indirizzo di base + X.**
CMPoper.,X	**$D5**	**Confronta l'accumulatore con la memoria. L'indirizzo è quello dell'operando + X.**

INDIRIZZAMENTO INDICIZZATO ASSOLUTO

L'indirizzamento indicizzato assoluto vi permette di accedere ad ogni locazione di memoria nei due banchi da 64 Kbyte. L'indirizzo effettivo è calcolato aggiungendo al byte basso dell'operando il contenuto di X o Y e prendendo poi il nuovo indirizzo a 16 bit. Dunque per il fatto di poter accedere a tutta la memoria questa istruzione richiederà 1 byte per l'op-code ed altri 2 per l'indirizzo.
Ecco alcuni esempi di istruzioni indicizzate assolute:

ISTRUZIONE	CODICE HEX	SIGNIFICATO
ANDoper.,Y	**$39**	**Esegue l'AND fra l'accumulatore e l'indirizzo calcolato aggiungendo X all'operando.**
ASLoper.,X	**$1E**	**Esegue lo scorrimento aritmetico a sinistra del contenuto dell'operando.**

IL CONCETTO DI INDIRIZZAMENTO INDIRETTO

Finora avete imparato che nel caso dell'indirizzamento indicizzato il computer calcola l'indirizzo effettivo prendendo quello di base ed aggiungendo uno spostamento dato dai registri indice. L'indirizzamento indiretto calcola diversamente l'indirizzo effettivo.
Pensate all'indirizzamento indiretto come all'indirizzo di un indirizzo. Ecco un esempio che usa l'indirizzamento indiretto:

```
JMP ($0326)
```

L'istruzione JMP qui sopra è un esempio di quanto detto. Essa richiede 3 byte: 1 per l'op-code e 2 per l'indirizzo. Le parentesi indicano che si sta usando l'indirizzamento indiretto. L'indirizzo fornito nell'istruzione indica il primo di due byte consecutivi in memoria ($0326-$0327): questi due byte contengono l'indirizzo effettivo a cui saltare, che non è quindi $0326, ma il contenuto dei due byte. Se quindi, per ipotesi, in $0326-$0327 fosse contenuto l'indirizzo $F265, l'esecuzione del programma riprenderebbe da $F265.

Se non fossero presenti le parentesi, l'assemblatore interpreterebbe l'istruzione come se fosse ad indirizzamento assoluto. Il processore prenderebbe quindi $0326 come indirizzo effettivo e riprenderebbe l'esecuzione da $0326. È ovvio che nell'indirizzamento indiretto si indica solo il primo byte dei due contenenti il vero indirizzo.
Gli ultimi due modi di indirizzamento, indicizzato indiretto ed indiretto indicizzato, usano lo stesso principio. Ecco la spiegazione per ognuno.

INDIRIZZAMENTO INDICIZZATO INDIRETTO

L'indirizzamento indicizzato indiretto è simile a quello assoluto indiretto, sebbene usi il registro X per modificare un indirizzo. Questo tipo di indirizzamento, spesso chiamato indirizzamento indiretto con X, richiede 2 byte: il primo per l'op-code ed il secondo per l'indirizzo di pagina zero da cui partire nel calcolo di quello effettivo. Il contenuto di X viene aggiunto all'indirizzo specificato, ignorando un eventuale riporto (l'indirizzo deve restare in pagina zero). Il risultato punta ancora a una locazione di pagina zero: questa e quella successiva contengono i byte basso/alto dell'indirizzo effettivo. Ecco un esempio:

```
LDX #$04
LDA #$00
STA ($DF,X)
```

La prima linea carica 4 in X. Poi, si mette 0 nell'accumulatore. La terza istruzione infine pone lo 0 all'indirizzo effettivo. Calcoliamo l'indirizzo effettivo prendendo l'indirizzo di base $DF (non il suo contenuto) e vi aggiunge il registro X (4), ottenendo $E3. I contenuti di $E3-E4 sono i byte basso/alto dell'indirizzo effettivo. Per esempio, se $E3-E4 contenessero $F356, l'indirizzo effettivo sarebbe $F356. L'indirizzamento indiretto indicizzato è anche chiamato pre-indicizzamento perchè avviene prima di ottenere l'indirizzo effettivo. L'indirizzamento indiretto con X è utile nell'indirizzare una serie di puntatori come la memoria di pagina zero del C128.

INDIRIZZAMENTO INDIRETTO INDICIZZATO

Questo modo, detto anche indirizzamento indiretto con Y, è post-indicizzato, il che significa che l'indirizzo effettivo si ottiene con la somma dell'indice stesso. Questo modo agisce sul principio di un indirizzo di base e un dislocamento. Ecco come lavora.
Il primo dei 2 byte è l'op-code, il secondo l'operando, che punta ad un indi-

rizzo di pagina zero. I contenuti di Y vengono aggiunti a quelli dell'indirizzo per formare il byte basso dell'indirizzo effettivo. Così i contenuti del puntatore fanno da indirizzo di base e quelli di Y da dislocamento. Nel caso vi sia un riporto, questo viene aggiunto alla locazione di memoria consecutiva al puntatore, e il cui contenuto diviene così il byte alto dell'indirizzo effettivo. Questa è la vera indicizzazione, appositamente progettata per manipolare tabelle di dati. Quindi, per accedere ai diversi valori della tabella, cambiate solo il valore di Y, poichè l'indirizzo di base è già stabilito: Ecco un esempio:

```
LDY #$08
LDA #$00
STA ($EA),Y
```

La prima istruzione carica 8 in Y. La seconda 0 in A. La terza pone il contenuto di A nell'indirizzo effettivo. Per trovare questo indirizzo, aggiungiamo il contenuto di Y (dislocamento) a quello della locazione puntata (indirizzo di base). Nell'esempio, $EA contiene $F0. Aggiungendo Y a $F0 arriviamo a $F8, il byte basso dell'indirizzo effettivo. Il byte alto si ottiene aggiungendo il carry (riporto, nessuno in questo caso) al contenuto del byte successivo a $EA. Per esempio, $EB contiene $3F. Poichè dunque il byte basso è $F8 e l'alto $3F, l'indirizzo effettivo sarà $3FF8.
Tenete conto della differenza fra i modi d'indirizzamento indicizzato indiretto ed indiretto indicizzato, poichè è facile confonderli. Ricordate, la differenza più importante è il modo in cui viene calcolato l'indirizzo effettivo. L'indicizzato indiretto è indicizzato con X, prima di arrivare all'indirizzo effettivo. L'indiretto indicizzato è post-indicizzato col registro Y.
Avete appena finito tutti i modi d'indirizzamento sul Commodore 128. Ognuno viene usato in circostanze differenti e dovete scegliere quello adeguato per ottenere le massime prestazioni dal microprocessore. Per esempio, usate l'indirizzamento di pagina zero al posto dell'indirizzamento assoluto quando state manipolando delle locazioni in pagina zero.

TIPI DI ISTRUZIONE

Questa sezione spiega tutti i tipi di istruzione del linguaggio macchina disponibili sul Commodore 128. Esse sono trattate prima per tipo di istruzione, come istruzioni REGISTRO/MEMORIA e di CONFRONTO; poi sono elencate alfabeticamente una per una con tutte le diverse opzioni di indirizzamento. Questa sezione fornisce informazioni importanti sulla programmazione in linguaggio macchina sul Commodore 128 (o qualunque altro microcomputer basato sul 6502).

Usate queste informazioni come un riferimento di fondo per ogni istruzione. La Figura 6-9 fornisce un elenco alfabetico dei mnemonici del processore 8502. Per informazioni più dettagliate e di veloce riferimento, andate alla sezione seguente dove c'è una lista in ordine alfabetico delle istruzioni, i loro op-code esadecimali, i differenti formati per ogni modo di indirizzamento e la loro influenza sui flag del registro di stato.

COMPENDIO DELLE ISTRUZIONI DEL MICROPROCESSORE 8502 IN ORDINE ALFABETICO

ADC Aggiunge la memoria e il carry all'accumulatore
AND AND fra memoria e accumulatore
ASL Scorrimento logico a sinistra

BCC Salta se carry 0
BCS Salta se carry 1
BEQ Salta se zero 1
BIT Test fra bit in memoria e nell'accumulatore
BMI Salta se negativo 1
BNE Salta se zero 0
BPL Salta se negativo 0
BRK Interruzione software
BVC Salta se eccesso 0
BVS Salta se eccesso 1

CLC Azzera il carry
CLD Azzera il decimale
CLI Azzera l'interruzione
CLV Azzera l'eccesso
CMP Confronta la memoria e l'accumulatore
CPX Confronta la memoria e X
CPY Confronta la memoria e Y

DEC Sottrae 1 dalla memoria
DEX Sottrae 1 da X
DEY Sottrae 1 da Y

EOR OR esclusivo fra la memoria e l'accumulatore

INC Aggiunge 1 alla memoria
INX Aggiunge 1 a X
INY Aggiunge 1 a Y

JMP Salta al nuovo indirizzo
JSR Chiama un indirizzo salvando l'indirizzo attuale

LDA Carica l'accumulatore dalla memoria
LDX Carica X dalla memoria
LDY Carica Y dalla memoria
LSR Scorrimento logico a destra

NOP Nessuna operazione

COMPENDIO DELLE ISTRUZIONI DEL MICROPROCESSORE 8502 IN ORDINE ALFABETICO

ORA	**OR fra l'accumulatore e la memoria**
PHA	**Pone l'accumulatore sullo stack**
PHP	**Pone il registro di stato sullo stacck**
PLA	**Preleva l'accumulatore dallo stack**
PLP	**Preleva il registro di stato dallo stack**
ROL	**Rotazione a sinistra**
ROR	**Rotazione a destra**
RTI	**Ritorno da interruzione**
RTS	**Ritorno da subroutine**
SBC	**Sottrae la memoria e il carry dall'accumulatore**
SEC	**Carry a 1**
SED	**Decimale a 1**
SEI	**Interruzione a 1**
STA	**Pone l'accumulatore in memoria**
STX	**Pone X in memoria**
STY	**Pone Y in memoria**
TAX	**Copia l'accumulatore in X**
TAY	**Copia l'accumulatore in Y**
TSX	**Copia il puntatore allo stack in X**
TXA	**Copia X in A**
TXS	**Copia X nel puntatore allo stack**
TYA	**Copia Y in A**

Figura 6-9. Mnemonici dell'8502

ISTRUZIONI REGISTRO/MEMORIA

Le istruzioni REGISTRO/MEMORIA sono:

```
LDA   STA
LDX   STX
LDY   STY
```

Le istruzioni registro/memoria piazzano un valore dalla memoria a un registro o viceversa.

CARICAMENTO DELL'ACCUMULATORE

La prima e più comune istruzione è LDA, Carica l'accumulatore. Questa pone un valore nell'accumulatore, il registro più potente ed usato del processore. Il valore da porre deriva dalla memoria o da una costante. Ecco un esempio:

LDA $2000

Questa istruzione carica il contenuto della locazione $2000 (8192) nell'accumulatore. Il valore in $2000 rimane uguale. Lo stesso valore rimarrà poi nell'accumulatore fino ad una nuova lettura od operazione che lo modifichi.
L'esempio precedente usa solo uno dei modi di indirizzamento per caricare l'accumulatore. Un'altra forma che possiamo usare è il caricamento di una costante. Per fare ciò, dovete precedere il segno del dollaro ($) col segno #. Per miglior leggibilità è consigliabile, ma non necessario, piazzare uno spazio fra l'op-code e l'operando. Ecco un esempio di caricamento di una costante nell'accumulatore:

LDA #$0A

Questo carica la costante $0A (10) nell'accumulatore. Ricordatevi di precedere una costante col segno #, altrimenti l'assemblatore interpreterà l'istruzione come se leggesse dalla pagina zero.
Le istruzioni LDXe LDY funzionano allo stesso modo di LDA. Ancora, potete caricare una costante o i contenuti della memoria nei registri X e Y. Esempi:

```
LDX #$0A
LDX $2000
LDX #$FB
```

PORRE: L'OPPOSTO DI CARICARE

Sapete come porre un valore in un registro, ma come fare l'opposto? L'istruzione STORE (poni) esegue il contrario di LOAD (carica). Piazza un valore dai registri A (accumulatore), X o Y in un indirizzo di memoria specificato. Come avete imparato nella sezione dell'indirizzamento, le istruzioni di store, load e molte altre hanno parecchi formati, dipendenti dal tipo di indirizzamento usato. Ecco un esempio:

STA $FC3E

Questo pone i contenuti dell'accumulatore nella locazione $FC3E, senza modificarli fino ad una nuova istruzione che li cambi specificamente. Le istruzioni STX e STY lavorano allo stesso modo; pongono i contenuti dei registri nella locazione di memoria specificata. Non esiste una versione in modo immediato dei comandi di store.

ISTRUZIONI DI CONTATORE

Le istruzioni di contatore sono:

```
INC   DEC
INX   DEX
INY   DEY
```

Le istruzioni di contatore possono essere usate per conservare una traccia o il numero di volte di un evento. Queste istruzioni sono usate per operazioni

matematiche o per indicizzare una serie di indirizzi. L'istruzione di incremento, INC, aggiunge 1 al contenuto di una locazione di memoria. Queste istruzioni sono usate principalmente in un ciclo di programma e in congiunzione con un'istruzione di diramazione. Ecco un esempio di ciclo e di come INC tenga conto dei giri eseguiti:

```
        LDX #$00
        TXA
START   STA $2000,X
        INX
        BNE START
```

La prima istruzione carica 0 in X. La seconda copia X in A, senza modificare X. La terza istruzione pone il contenuto di A (0) nella locazione $2000 + X, cioè $2000 la prima volta. La quarta istruzione incrementa X, e la quinta salta a START finchè X non è 0. Questo segmento di programma pone 0 in un'intera pagina (256 locazioni) iniziando a $2000 e finendo a $20FF. Quando X raggiunge 255, il nuovo incremento lo fa passare a 0 e l'istruzione di salto non è più valida; il programma esce così dal ciclo.
L'istruzione INY opera allo stesso modo di INX, poichè anch'essa usa l'indirizzamento implicito. L'istruzione INC, al contrario, usa parecchi modi d'indirizzamento compreso l'assoluto, che usa indirizzi a 16 bit. Con l'istruzione INC, potete contare oltre la capacità di un numero a 8 bit, sebbene dobbiate separare il contatore in byte basso e alto. Per esempio, il byte basso conta gli incrementi minori di una pagina, mentre quello alto conta il numero delle pagine. Quando il contatore del byte basso arriva a 255 e viene incrementato, esso si azzera. Quando ciò accade, incrementate il byte alto del contatore. Per contare fino a 260 (decimale), il byte alto equivale a 1 e il basso a 4. Ecco un'equazione per illustrare il concetto:

$$(1 * 256) + 4 = 260$$

Ecco il codice macchina che lo mette in atto:

```
        LDA #$00
        STA ALTO
        STA BASSO
LOOP    INC BASSO
        BNE LOOP
        INC ALTO
LOOP2   INC BASSO
        LDA BASSO
        CMP #$04
        BNE LOOP2
```

Le istruzioni di decremento operano allo stesso modo di quelle di incremento. Sono la controparte a numeri negativi dei contatori ad incremento.

ISTRUZIONI DI CONFRONTO

Il Commodore 128 ha tre istruzioni di confronto che comparano i contenuti di un registro con quelli della memoria. Un'istruzione di confronto può essere usata per determinare quale istruzione eseguire in base al valore di una condizione. Le istruzioni di confronto sono:

```
CMP
CPX
CPY
```

L'istruzione CMP confronta i contenuti dell'accumulatore con quelli dell'indirizzo specificato nell'istruzione. Essenzialmente le istruzioni di confronto sottraggono il contenuto della memoria da quello del registro, ma non cambiano nessuno dei due - condizionano solo i flag. CPX confronta i contenuti di X con quelli dell'indirizzo specificato, e CPY i contenuti di Y.
Tutte e tre le istruzioni hanno versioni operanti nei modi di indirizzamento immediato, pagina zero e assoluto. Ciò significa che potete confrontare i contenuti di un registro (A, X o Y) con quelli di una locazione di pagina zero o superiore, od anche con una costante. Ecco un esempio:

```
        LDX #$00
        LDA #$00
ONE     STA $DF,X
        INX
        CPX #$0A
        BNE ONE
```

Il segmento di programma precedente pone degli zero in 10 locazioni di memoria consecutive, partendo da $DF. La prima istruzione carica 0 in X, e la seconda 0 nell'accumulatore. La terza istruzione pone 0 nella locazione $DF+X. La quarta istruzione incrementa X. La quinta confronta i contenuti di X con la costante 10 ($0A). Se X non è uguale a 10, il programma salta indietro a ONE. Dopo che il ciclo ha girato 10 volte, il registro X vale 10. Quindi il processore non esegue il salto e prosegue con l'istruzione dopo BNE.
Potete confrontare il valore di un registro con quello di una locazione di memoria. Ecco lo stesso esempio di prima usando i contenuti di una locazione di memoria piuttosto che una costante:

```
        LDA #$0A
        STA $FB
        LDX #$00
        LDA #$00
ONE     STA $DF,X
        INX
        CPX $FB
        BNE ONE
```

Ricordate, se volete confrontare numeri maggiori di un byte (più di 255) dovete separare il numero in byte basso e alto.
Pure l'istruzione BIT può essere usata nei confronti. Guardate le istruzioni logiche qui vicino.

ISTRUZIONI ARITMETICHE E LOGICHE

L'accumulatore è responsabile di tutte le operazioni matematiche e logiche che il vostro computer esegue. Le istruzioni logiche e matematiche disponibili in linguaggio macchina sono:

```
ADC   EOR
AND   ORA
BIT   SBC
```

Ecco il significato di ogni istruzione:

ADC – Aggiunge i contenuti di una specifica locazione di memoria e il carry all'accumulatore. É considerata una buona pratica di programmazione azzerare il carry (CLC) prima di ogni addizione. Questo per evitare di aggiungere il carry in ogni addizione.
AND – Esegue l'operazione logica AND fra i contenuti dell'accumulatore e quelli dell'indirizzo di memoria specificato, lasciando il risultato nell'accumulatore.
BIT – Confronta i bit della locazione di memoria specificata con quelli dell'accumulatore, dando un risultato zero/non zero. Inoltre i bit 6 e 7 della locazione vengono ricopiati nei bit 6 (eccesso) e 7 (negativo) del registro di stato.
EOR – Esegue l'OR esclusivo fra la locazione di memoria e l'accumulatore.
ORA – Esegue l'OR logico fra la locazione di memoria e l'accumulatore.
SBC – Sottrae i contenuti della locazione di memoria specificata e il carry dall'accumulatore. É buona pratica mettere a 1 il carry prima di eseguire la sottrazione, così da evitare di sottrarlo dall'accumulatore (per sottrarre l'operando, e pure il carry, viene eseguita una complementazione e poi un'addizione; aggiungere il contrario infatti equivale a sottrarre).

ISTRUZIONI ARITMETICHE (ADC, SBC)

Le istruzioni di addizioni e sottrazione sono facili da capire. Ecco un esempio:

```
CLC
LDA #$0A
STA $FB
ADC #$04
SEC
SBC #$06
ADC $FB
STA $FD
```

Questo segmento di programma essenzialmente esegue le istruzioni matematiche seguenti: (10+4)-6+10=18.
La prima istruzione azzera il carry, e la seconda carica 10 in A. La terza istruzione pone il valore di A (10) in $FB per usarlo più tardi. La quarta istruzione aggiunge 4 al contenuto di A (10). L'istruzione SBC sottrae invece 6 da A. L'istruzione seguente, ADC $FB, aggiunge il contenuto di $FB (10) all'accumulatore. Il valore risultante (18) viene poi posto in $FD.

ISTRUZIONI LOGICHE (AND, EOR e ORA)

Queste istruzioni operano sui contenuti di un indirizzo di memoria o di un registro. L'operazione AND è un'operazione di algebra binario (Booleano) a due operandi, che può risultare in due valori, 0 o 1. Il solo modo di ottenere un 1 è avere entrambi gli operandi a 1; altrimenti il risultato è 0. Per esempio, i due operandi sono il contenuto di uno specifico indirizzo di memoria e dell'accumulatore. Ecco una spiegazione di questo concetto:

Indirizzo di memoria	**= 10101010**
Accumulatore	**= 10000011**
Risultato di AND	**= 10000010**

Come si può notare, il risultato di un'operazione AND è 1 (vero), solo se i due operandi sono uguali a 1; altrimenti risulta uno 0. Notate che il bit 7 (il bit più significativo) vale 1 perchè entrambi i bit degli operandi sono a 1. L'unico altro bit a 1 è il bit 1, poichè entrambi i bit 1 sono a 1. Il resto dei bit è a 0 perchè nessuna posizione di bit in entrambi gli operandi è a 1. Un 1 e uno 0 equivalgono a 0, come uno 0 e uno 0.
L'OR Booleano lavora differentemente. La regola generale è:

Se uno degli operandi è a 1, il valore Booleano risultante equivale a 1. Per esempio, i due operandi sono il contenuto di uno specifico indirizzo di memoria e dell'accumulatore. Ogni bit può essere trattato individualmente come un operando. Ecco un esempio illustrato.

Indirizzo di memoria	**= 10101001**
Accumulatore	**= 10000011**
Risultato di OR	**= 10101011**

Per tutte le posizioni di bit che sono a 1 in uno degli operandi, il valore risultante è 1. Il risultato è 1 se uno o entrambi gli operandi sono a 1. L'OR esclusivo lavora similmente all'operazione OR, eccettuato che se entrambi gli operandi sono a 1, il risultato è 0. Ciò suggerisce la seguente regola generale:

Se uno dei due operandi è a 1, il valore Booleano risultante è 1; se entrambi sono a 1 il valore è 0.

Ecco un esempio che usa questa regola:

Indirizzo di memoria	**= 10101001**
Accumulatore	**= 10000011**
Risultato di OR	**= 00101010**

In questo esempio, gli operandi sono gli stessi del precedente OR. Notate che i bit 7 e 0 sono ora a 0 poichè entrambi gli operandi sono a 1. Tutti gli altri valori dei bit rimangono gli stessi.

BIT

L'istruzione BIT esegue un'operazione logica AND fra i contenuti di una locazione di memoria e quelli dell'accumulatore, senza conservare il risultato. Invece, viene influenzato il flag di zero. Questa istruzione fa un confronto bit a bit dei contenuti dell'accumulatore e della locazione di memoria. Se il risultato ha tutti i bit a 0, allora il flag Z va a 1, altrimenti viene posto a 0.
Il vostro programma in linguaggio macchina può quindi agire in dipendenza dal flag di zero. Inoltre, i bit 7 e 6 della locazione di memoria sono ricopiati rispettivamente nei flag di negativo ed eccesso del registro di stato. Pure questi flag possono essere usati per eseguire delle istruzioni a seconda del loro stato. Per esempio, l'istruzione BIT esegue:

Locazione di memoria	**= 10101001**
Accumulatore	**= 11001101**
Risultato di BIT (non conservato)	**= 10001001**
Registro di stato	**= 10 0**
	NV BDIZC
	7 0

Poichè la maschera di bit risultante è diversa da 0, il flag zero del registro di stato non è a 1. Inoltre, i bit 7 e 6 sono piazzati rispettivamente nei flag di negativo ed eccesso del registro. Notate che il risultato dell'AND non viene conservato, e i contenuti dell'accumulatore e della locazione di memoria sono lasciati inalterati. Osservate l'esempio seguente di due maschere di bit che risultano in 0 in un AND:

Locazione di memoria	**= 01111010**
Accumulatore	**= 10000100**
Risultato di BIT (non conservato)	**= 00000000**
Registro di stato	**= 01 1**
	NV BDIZC
	7 0

Questa volta la maschera di bit risulta a 0. Perciò il flag di zero del registro di stato è posto a 1. I bit 7 e 6 sono ricopiati rispettivamente nei flag di negativo ed eccesso.
Ora sapete il modo di operare di ognuna delle istruzioni logiche ed aritmetiche. La prossima sezione tratta le istruzioni di diramazione. Queste istruzioni sono disegnate appositamente per eseguire un certo insieme di istruzioni a seconda di una condizione. Molte volte queste condizioni si riferiscono al risultato di operazioni aritmetico-logiche, le quali condizionano i flag del registro di stato. Le istruzioni di diramazione agiscono quindi in accordo a questi flag.

ISTRUZIONI DI DIRAMAZIONE

Il microprocessore 8502 ha molte istruzioni di salto condizionato. Per definizione, una diramazione ridirige di volta in volta il flusso, altrimenti sequenziale, di un programma. Essa trasferisce il controllo ad un'istruzione diversa da quella immediatamente successiva.
Le istruzioni di salto condizionato fanno in modo che il processore esamini un flag particolare del registro di stato. Il processore, a seconda del valore del flag controllato, può sia eseguire il salto sia proseguire con l'istruzione successiva a quella di diramazione.
Pensate al salto condizionato come a un test. Per esempio, se la condizione è vera, il programma salta o sposta il controllo ad un'istruzione diversa da quella immediatamente successiva. Se il test fallisce, l'esecuzione del programma riprende dall'istruzione successiva a quella di salto. Ricordate che il controllo del programma può essere trasferito da un'altra parte pure se il test fallisce. Ciò significa che potete trasferire il controllo del programma a seconda della condizione stabilita. Potete quindi saltare se un flag vale 0 o viceversa se vale 1.

Le istruzioni di salto condizionato disponibili sull'8502 sono:

```
BCC    BNE
BCS    BPL
BEQ    BVC
BMI    BVS
```

Ecco il significato di ognuna delle istruzioni di salto. Le frasi fra parentesi sono la trascrizione letterale dei mnemonici. Il resto spiega il significato di ogni opcode.

BCC — Branch on Carry Clear: Salto se carry a 0) Salta se il carry vale 0.
BCS — (Branch on Carry Set: Salto se carry a 1) Salta se il carry vale 1.
BEQ — (Branch on result EQ zero: salto se risultato 0) Salta se il flag di zero vale 1.
BMI — (Branch on result MInus: salto se negativo a 1) Salta se il flag di negativo vale 1.
BNE — (Branch on result Not Equal to zero: salto se risultato diverso da 0) Salta se il flag di zero vale 0.
BPL — (Branch on result PLus: salto se risultato positivo) Salta se il flag di negativo vale 0.
BVC — (Branch on oVerflow Clear: salto se il risultato non eccede 7 bit) Salta se il flag di eccesso vale 0.
BVS — (Branch on oVerflow Set: salto se il risultato eccede i 7 bit) Salta se il flag di eccesso vale 1.

Come potete vedere, tutte le istruzioni di salto dipendono dal valore dei flag del registro di stato.

Ecco alcuni esempi di diramazione.

```
READY.

MONITOR
    PC   SR  AC  XR  YR  SP
; FB000  00  00  00  00  F8



. 01828  E6  FA    INC  $FA
. 0182A  A5  FA    LDA  $FA
. 0182C  D0  02    BNE  $1830
. 0182E  E6  FB    INC  $FB
. 01830  C8        INY
```

Questo segmento di programma tiene conto dei byte basso/alto del puntatore in $FA-FB. La prima istruzione (INC $FA) incrementa il byte basso. Poi, l'accumulatore viene caricato da $FA. L'istruzione di salto (BNE $1830) valuta il contenuto dell'accumulatore. Se questo valore non è zero, avviene il salto alla locazione $1830 (INY). In questo caso non si deve ancora incrementare il byte alto del puntatore, così l'istruzione INC $FB viene saltata. Se invece il valore dell'accumulatore fosse zero, allora il salto non sarebbe eseguito e si incrementerebbe il byte alto (INC $FB).

Questo è un esempio dell'istruzione BPL (Branch on result PLus).

```
READY.

MONITOR
    PC   SR  AC  XR  YR  SP
; FB000  00  00  00  00  F8
. 01858  8E  00  D6  STX  $D600
. 0185B  2C  00  D6  BIT  $D600
. 0185E  10  FB      BPL  $185B
. 01860  8D  01  D6  STA  $D601
```

Questo esempio è una routine che controlla il bit di stato del registro indirizzo dell'8563, per sapere se il bus dei dati è disponibile. La prima istruzione memorizza nel registro indirizzo dell'8563 il contenuto del registro X, che è

stato precedentemente caricato col numero di un registro interno dell'8563. L'istruzione BIT copia il bit 7 di $D600 nel flag di negativo del registro di stato dell'8502. L'istruzione BPL salta all'istruzione BIT in $185B finchè il valore del flag di negativo è 0. Per il chip 8563, il bit 7 a 0 significa che il bus dei dati non è ancora pronto per letture o scritture. Il ciclo continua finchè il bit 7 non è a 1: ora il risultato sarà negativo, così la diramazione non verrà più eseguita e si passerà il controllo all'istruzione immediatamente successiva, che pone il dato nel registro dati dell'8563.

ISTRUZIONI DI TRASFERIMENTO FRA REGISTRI

Le istruzioni di trasferimento fra registri copiano un valore da un registro (A, X o Y) in un altro. Queste istruzioni sono utili poichè richiedono solo un byte di memoria ed evitano di utilizzare una locazione intermedia in cui porre il valore da caricare poi nell'altro registro. Il processore 8502 ha le seguenti sei istruzioni di trasferimento fra registri:

TAX — Copia l'accumulatore in X.
TAY — Copia l'accumulatore in Y.
TSX — Copia il puntatore allo stack in X.
TXA — Copia X nell'accumulatore.
TYA — Copia Y nell'accumulatore.
TXS — Copia X nel puntatore dello stack.

Le istruzioni TXS e TSX copiano valori dal registro X al puntatore allo stack e viceversa. Questo è utile in un'operazione matematica per prelevare un valore dallo stack di volta in volta (per esempio, eseguire un calcolo su di esso e porre il risultato al suo posto nello stack). Un altro uso consiste nel recuperare un valore dallo stack, salvarlo nel registro Y, porre un nuovo valore al suo posto ed infine porvi sopra il vecchio valore. Questo potrebbe essere il caso di un ordinamento ascendente.

ISTRUZIONI DI SCORRIMENTO E ROTAZIONE

Le istruzioni di scorrimento e rotazione manipolano i bit dell'accumulatore o della memoria. Qui di seguito ecco le istruzioni usate dalla famiglia dell'8502:

ASL — Scorrimento aritmetico a sinistra
LSR — scorrimento logico a destra
ROL — Rotazione a sinistra
ROR — Rotazione a destra

ISTRUZIONI DI SCORRIMENTO

Le istruzioni di scorrimento sono utili nella valutazione di singoli bit di una sequenza di controllo del programma. Per esempio, una routine di lettura del joystick può far uso delle istruzioni di scorrimento. Le locazioni $DC00 e $DC01 riportano la direzione del joystick (bit 0-3) e lo stato del pulsante di fuoco

(bit 4). Un modo di valutare i singoli bit è di farli scorrere a destra. Ciò fa sì che ogni singolo bit passi nel carry. Se il carry vale 1, allora il joystick è premuto nella direzione corrispondente. Ecco una routine di lettura del joystick che usa l'istruzione LSR per valutarne la direzione:

```
READY.
MONITOR
    PC  SR  AC  XR  YR  SP
; FB000  00  00  00  00  F8

. 01800  AD  00  DC   LDA  $DC00
. 01803  A0  00       LDY  #$00
. 01805  A2  00       LDX  #$00
. 01807  4A           LSR
. 01808  B0  01       BCS $180B
. 0180A  88           DEY
. 0180B  4A           LSR
. 0180C  B0  01       BCS $180F
. 0180E  C8           INY
. 0180F  4A           LSR
. 01810  B0  01       BCS $1813
. 01812  CA           DEX
. 01813  4A           LSR
. 01814  B0  01       BCS $1817
. 01816  E8           INX
. 01817  4A           LSR
. 01818  86  FA       STX $FA
. 0181A  84  FB       STY $FB
. 0181C  60           RTS
```

ISTRUZIONI DI ROTAZIONE

Le istruzioni di rotazione operano in modo leggermente differente. Anche qui un bit viene fatto cadere nel carry, ma all'estremità opposta, anzichè uno 0 entra il contenuto precedente del carry. Si realizza così una rotazione a 9 bit. Nel caso di ROR : 1) il byte scorre di una posizione a destra; 2) contemporaneamente il contenuto del carry va ad occupare il bit 7, mentre il bit 0 entra nel carry.

Figura 6-10. Concetto di ROR. Il concetto di ROL è esattamente l'opposto.

ISTRUZIONI DI SET E CLEAR (PORRE A 1 E AZZERARE)

Le istruzioni di set e clear sono progettate per manipolare i flag del registro di stato e controllare certe condizioni del processore. Queste sono le istruzioni disponibili sull'8502:

SEC - Carry a 1.
SED - Decimale a 1.

SEI - Interruzione a 1.
CLC - Carry a 0.
CLD - Decimale a 0.
CLI - Interruzione a 0.
CLV - Eccesso a 0.

Ognuna di queste istruzioni riguarda un determinato flag del registro di stato. Notate come ogni istruzione di clear abbia una controparte per porre a 1, eccetto CLV (CLear oVerflow flag: azzera l'eccesso). Il flag di eccesso può essere messo a 1 da un'istruzione BIT oppure da un'operazione matematica in cui il segno dell'operando è stato cambiato indebitamente.

La figura 6-11 mostra il registro di stato dell'8502:

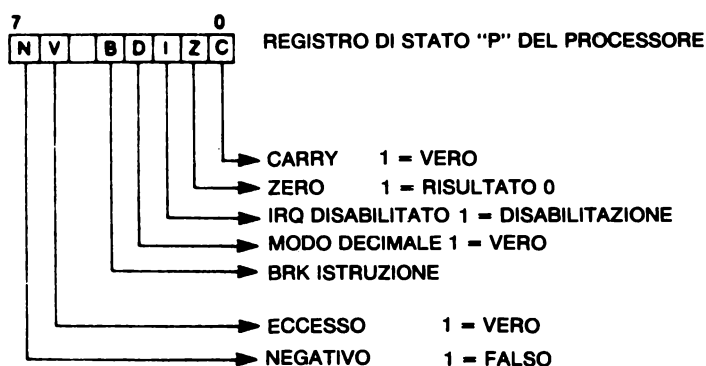



Figura 6-11. Registro di stato dell'8502

I flag del registro di stato vanno a 1 per varie ragioni. Per esempio, entrate in modo decimale quando volete eseguire dei calcoli in notazione BCD (binario codificato decimale) piuttosto che in esadecimale. Ponete a 1 il carry per eseguire una sottrazione. Disabilitate le interruzioni ponendo a 1 il flag I. Un esempio di routine di scroll fine di una parte dello schermo, funzionante a interruzioni, è dato alla fine del Capitolo 9, Vol. III.

Le istruzioni di clear funzionano nel modo opposto. Per assicurarvi di non aggiungere il carry in un'addizione, azzeratelo con CLC. Per eseguire operazio-

ni matematiche in esadecimale o binario, azzerate il flag di decimale così che i vostri calcoli non siano erroneamente interpretati in BCD. Quando il risultato di un'operazione con segno eccede i 7 bit, lo stato del bit di segno viene cambiato. Questo è un eccesso, e viene segnalato dal flag V. Per correggerlo, cambiate di nuovo lo stato del segno e poi azzerate il flag V: questo flag infatti può passare da 0 a 1 ma non viceversa, se non usate le istruzioni CLV e BIT.
Quando un programma risponde a un'interruzione, per prima cosa ponete a 1 il flag d'interruzione, per evitare di rispondere ad altre interruzioni. Poi, una volta eseguita la routine di servizio, riabilitate le interruzioni con CLI.

ISTRUZIONI DI SALTO E RITORNO

ISTRUZIONI DI SALTO

Il processore 8502 fa uso di due istruzioni di salto:

JMP — Salto incondizionato a nuova locazione.
JSR — Salto incondizionato a nuova locazione salvando l'indirizzo di ritorno.

Entrambe queste istruzioni ridirigono il controllo del programma ad una locazione diversa da quella immediatamente successiva. La prima istruzione, JMP, è un salto di sola andata alla locazione specificata nel campo operando, o al contenuto di esso (indiretto). Per esempio:

```
JMP $1800
```

salta alla locazione $1800 ed esegue l'istruzione contenutavi. Questo è un salto diretto. Potete anche saltare indirettamente. Per esempio:

```
JMP ($1800)
```

salta all'indirizzo contenuto nelle locazioni $1800-1801. Per ipotesi, $1800 contiene il valore $FE e $1801 il valore $C0. Quindi, l'istruzione sopra salta a $C0FE, e non $1800. Il salto indiretto è sempre identificato dalle parentesi che racchiudono l'operando, e significa di saltare all'indirizzo contenuto nell'operando.
L'istruzione JSR chiama delle subroutine salvando l'indirizzo di ritorno sullo stack. Così, quando si incontra l'istruzione RTS alla fine della subroutine, il processore preleva dallo stack l'indirizzo di ritorno e riprende l'esecuzione del programma principale dall'istruzione successiva a JSR. In breve, JSR è un salto ad andata e ritorno, mentre JMP è a sola andata. Per esempio:

```
.    01804  20  58  18  JSR  $1858
.    01807  A2  0C      LDX  #$0C
```

salta alla subroutine che inizia a $1858. L'indirizzo di ritorno viene salvato sullo stack, così quando nella subroutine viene incontrata l'istruzione RTS:

```
. 01858  8E  00  D6  STX  $D600
. 0185B  2C  00  D6  BIT  $D600
. 0185E  10  FB      BPL  $185B
. 01860  8D  01  D6  STA  $D601
. 01863  60          RTS
```

il processore riprende dal programma principale con l'istruzione LDX #$0C nella locazione $1807.

ISTRUZIONI DI RITORNO

Il set di istruzioni dell'8502 ha due istruzioni di ritorno:

RTI — Ritorno da interruzione.
RTS — Ritorno da subroutine

La prima istruzione ritorna dalla vostra routine di gestione dell'interruzione e dev'essere l'ultima istruzione del programma. La routine di servizio dell'interruzione è la serie di istruzioni eseguite in risposta ad un'interruzione. Riferitevi al Capitolo 9 (Vol. III), Programma di scroll fine di parte dello schermo funzionante col raster interrupt, per un esempio reale di routine di servizio.
L'istruzione RTS è l'ultima istruzione ad essere eseguita in una subroutine chiamata dal BASIC con SYS o dal linguaggio macchina con JSR. Guardate le istruzioni di salto più sopra per un esempio.

ISTRUZIONI DI STACK

Il set dell'8502 include 4 istruzioni di manipolazione dello stack. Eccole:

PHA — Porre l'accumulatore sullo stack.
PHP — Porre il registro di stato sullo stack.
PLA — Recuperare l'accumulatore dallo stack.
PLP — Recuperare il registro di stato dallo stack.

Il termine "Porre" significa piazzare un valore in cima allo stack, mentre "Recuperare" rimuovere questo valore dallo stack. I soli valori che si possono porre e recuperare sono l'accumulatore e il registro di stato. La manipolazione dei valori dello stack è importante per il programmatore che usa le interruzioni. Il Programma di scroll fine di parte dello schermo mediante raster interrupt, nel Capitolo 9 (Vol. III), mostra la manipolazione dei valori nello stack prima di ritornare da un'interruzione.

L'ISTRUZIONE NOP

L'istruzione NOP significa Nessuna OPerazione. É usata spesso per separare dei segmenti di programma per leggibilità o per riservare spazio per correzioni future. Questa istruzione non ha nessun effetto.

TAVOLA DELLE ISTRUZIONI E DEI MODI D'INDIRIZZAMENTO DELL'8502

Le prossime pagine contengono la Tavola delle Istruzioni e dei Modi d'Indirizzamento dell'8502. Queste sono le convenzioni usate nella tavola:

1. OP-CODE
2. Breve definizione
3. Notazione dell'operazione
4. Flag di stato
5. Flag influenzati
6. Modi d'indirizzamento
7. Formato in assembly
8. OP-CODE (in esadecimale)
9. Numero di byte
10. Numero di cicli dell'istruzione

In questo sommario sono applicate le seguenti convenzioni:

A	**Accumulatore**
X,Y	**Registri indice**
M	**Memoria**
P	**Registro di stato**
S	**Puntatore allo stack**
√	**Cambiamento**
-	**Nessun cambiamento**
+	**Somma**
∧	**AND logico**
-	**Sottrazione**
∀	**OR esclusivo logico**
↑	**Recupero dallo stack**
↓	**Porre sullo stack**
→	**Trasferisce a**
←	**Trasferisce da**
V	**OR logico**
PC	**Contatore di programma**
PCH	**Contatore di programma alto**
PCL	**Contatore di programma basso**
OP	**Operando**
#	**Immediato**

ADC Somma la memoria all'accumulatore con riporto

Operazione: A + M + C → A, C

N	Z	C	I	D	V
√	√	√	-	-	√

Indirizzam.	Assembly	Code	Byte	Cicli
Immediato	ADC #Op	69	2	2
Pag. zero	ADC Op	65	2	3
Pag. zero,X	ADC Op,X	75	2	4
Assoluto	ADC Op	6D	3	4
Assoluto,X	ADC Op,X	7D	3	4*
Assoluto,Y	ADC Op,Y	79	3	4*
(Ind.,X)	ADC (Op,X)	61	2	6
(Ind.),Y	ADC (Op),Y	71	2	5*

* Aggiungere 1 in caso di supero pagina

AND AND fra memoria e accumulatore

Operazione: A ∧ M → A

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Immediato	AND ×Op	29	2	2
Pag. zero	AND Op	25	2	3
Pag. zero,X	AND Op,X	35	2	4
Assoluto	AND Op	2D	3	4
Assoluto,X	AND Op,X	3D	3	4*
Assoluto,Y	AND Op,Y	39	3	4*
(Ind.,X)	AND (Op,X)	21	2	6
(Ind.),Y	AND (Op),Y	31	2	5

* Aggiungere 1 in caso di supero pagina

ASL Scorrimento aritmetico a sinistra

Operazione:
C ← 7 6 5 4 3 2 1 0 0 ← 0

N	Z	C	I	D	V
√	√	√	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Accumul.	ASL A	0A	1	2
Pag. zero	ASL Op	06	2	5
Pag. zero,X	ASL Op,X	16	2	6
Assoluto	ASL Op	0E	3	6
Assoluto, X I	ASL Op,X	1E	3	7

BCC Salta se carry 0

Operazione: Salto se C = 0 **N Z C I D V**

– – – – – –

Indirizzam.	Assembly	Code	Byte	Cicli
Relativo	BCC Op	90	2	2*

* Aggiungere 1 in caso di salto
* Aggiungere 2 in caso di supero pagina

BCS Salta se carry 1

Operazione: Salto se C = 1 **N Z C I D V**

– – – – – –

Indirizzam.	Assembly	Code	Byte	Cicli
Relativo	BCS Op	B0	2	2*

* Aggiungere 1 in caso di salto
* Aggiungere 2 in caso di supero pagina

BEQ Salta se zero 1

Operazione: Salto se Z = 1 **N Z C I D V**

– – – – – –

Indirizzam.	Assembly	Code	Byte	Cicli
Relativo	BEQ Op	F0	2	2*

* Aggiungere 1 in caso di salto
* Aggiungere 2 in caso di supero pagina

BIT Test fra bit e accumulatore

Operazione: A → M, M_7 → N, M_6 → V **MN Z✓ C I D M_6**

Indirizzam.	Assembly	Code	Byte	Cicli
Pag. zero	BIT Op	24	2	3
Assoluto	BIT Op	2C	3	4

BMI Salta se negativo

Operazione: Salto se N = 1

N	Z	C	I	D	V
–	–	–	–	–	–

Indirizzam.	Assembly	Code	Byte	Cicli
Relativo	BMI Oper	30	2	2*

* Aggiungere 1 in caso di salto
* Aggiungere 2 in caso di supero pagina

BNE Salta se risultato non zero

Operazione: Salto se Z = 0

N	Z	C	I	D	V
–	–	–	–	–	–

Indirizzam.	Assembly	Code	Byte	Cicli
Relativo	BNE Op	D0	2	2*

* Aggiungere 1 in caso di salto
* Aggiungere 2 in caso di supero pagina

BPL Salta se positivo

Operazione: Salto se N = 0

N	Z	C	I	D	V
–	–	–	1	–	–

Indirizzam.	Assembly	Code	Byte	Cicli
Relativo	BPL Op	10	2	2*

* Aggiungere 1 in caso di salto
* Aggiungere 2 in caso di supero pagina

BRK Forza un'interruzione

Operazione: PC + 2 ↓ P ↓

N	Z	C	I	D	V
–	–	–	1	–	–

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	BRK	00	1	7

1. Un comando BRK agisce anche con I = 1

BVC Salta se non eccesso

Operazione: Salto se V = 0

N	Z	C	I	D	V
-	-	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Relativo	BVC Op	50	2	2*

* Aggiungere 1 in caso di salto
* Aggiungere 2 in caso di supero pagina

BVS Salta se eccesso

Salto se V = 1

N	Z	C	I	D	V
-	-	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Relativo	BVS Op	70	2	2*

* Aggiungere 1 in caso di salto
* Aggiungere 2 in caso di supero pagina

CLC Azzera il carry

Operazione: 0 → C

N	Z	C	I	D	V
-	-	0	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	CLC	18	1	2

CLD Azzera il decimale

Operazione: 0 → D

N	Z	C	I	D	V
-	-	-	-	0	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	CLD	D8	1	2

CLI Azzera l'interruzione

Operazione: 0 → I

N	Z	C	I	D	V
-	-	-	0	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	CLI	58	1	2

CLV Azzera l'eccesso

Operazione: 0 → V

N	Z	C	I	D	V
-	-	-	-	-	0

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	CLV	B8	1	2

CMP Confronta la memoria e l'accumulatore

Operazione: A - M

N	Z	C	I	D	V
√	√	√	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Immediato	CMP #Op	C9	2	2
Pag. zero	CMP Op	C5	2	3
Pag. zero,X	CMP Op,X	D5	2	4
Assoluto	CMP Op	CD	3	4
Assoluto,X	CMP Op,X	DD	3	4*
Assoluto,Y	CMP Op,Y	D9	3	4*
(Ind.,X)	CMP (Op,X)	C1	2	6
(Ind.),Y	CMP (Op),Y	D1	2	5*

* Aggiungere 1 per supero pagina

CPX Confronta la memoria e X

Operazione: X – M

N	Z	C	I	D	V
√	√	√	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Immediato	CPX #Op	E0	2	2
Pag. zero	CPX Op	E4	2	3
Assoluto	CPX Op	EC	3	4

CPY Confronta la memoria e Y

Operazione: Y – M

N	Z	C	I	D	V
√	√	√	-	-	-

Assembly	Code	Byte	Cicli	
Immediato	CPY #Op	C0	2	2
Pag. zero	CPY Op	C4	2	3
Assoluto	CPY Op	CC	3	4

DEC Sottrae 1 dalla memoria

Operazione: M – 1 → M

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Pag. zero	DEC Op	C6	2	5
Pag. zero,X	DEC Op,X	D6	2	6
Assoluto	DEC Op	CE	3	6
Assoluto,X	DEC Op,X	DE	3	7

DEX Sottrae 1 da X

Operazione: X – 1 → X

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	DEX	CA	1	2

DEY Sottrae 1 da Y

Operazione: Y – 1 → Y

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	DEY	88	1	2

EOR OR esclusivo fra la memoria e l'accumulatore

Operazione: A→∀ A

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Immediato	EOR #Op	49	2	2
Pag. zero	EOR Op	45	2	3
Pag. zero,X	EOR Op,X	55	2	4
Assoluto	EOR Op	4D	3	4
Assoluto,X	EOR Op,X	5D	3	4*
Assoluto,Y	EOR Op,Y	59	3	4*
(Ind.,X)	EOR (Op,X)	41	2	6
(Ind.),Y	EOR (Op),Y	51	2	5*

* Aggiungere 1 per supero pagina

INC Aggiunge 1 alla memoria

Operazione: M + 1 → M

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Pag. zero	INC Op	E6	2	5
Pag. zero,X	INC Op,X	F6	2	6
Assoluto	INC Op	EE	3	6
Assoluto,X	INC Op,X	FE	3	7

INX Aggiunge 1 a X

Operazione: X + 1 → X

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	INX	E8	1	2

INY Aggiunge 1 a Y

Operazione: Y + 1 → Y

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	INY	C8	1	2

JMP Salta a nuova locazione

Operazione: (PC + 1) → PCL
(PC + 2) → PCH

N	Z	C	I	D	V
-	-	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Assoluto	JMP Op	4C	3	3
Indiretto	JMP (Op)	6C	3	5

JSR Salta a nuova locazione salvando l'indirizzo di ritorno

Operazione: PC+2↓,(PC+1)→PCL
(PC+2)→PCH

N	Z	C	I	D	V
-	-	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Assoluto	JSR Op	20	3	6

LDA Carica l'accumulatore dalla memoria

Operazione: M→A

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Immediato	LDA #Op	A9	2	2
Pag. zero	LDA Op	A5	2	3
Pag. zero,X	LDA Op,X	B5	2	4
Assoluto	LDA Op	AD	3	4
Assoluto,X	LDA Op,X	BD	3	4*
Assoluto,Y	LDA Op,Y	B9	3	4*
(Ind.,X)	LDA (Op,X)	A1	2	6
(Ind.),Y	LDA (Op),Y	B1	2	5*

* Aggiungere 1 per supero pagina

LDX Carica il registro X dalla memoria

Operazione: M→X

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Immediato	LDX #Op	A2	2	2
Pag. zero	LDX Op	A6	2	3
Pag. zero,Y	LDX Op,Y	B6	2	4
Assoluto	LDX Op	AE	3	4
Assoluto,Y	LDX Op,Y	BE	3	4*

* Aggiungere 1 per supero pagina

LDY Carica il registro Y dalla memoria

Operazione: M→Y

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Immediato	LDY ×Op	A0	2	2
Pag. zero	LDY Op	A4	2	3
Pag. zero,X	LDY Op,X	B4	2	4
Assoluto	LDY Op	AC	3	4
Assoluto,X	LDY Op,X	BC	3	4*

* Aggiungere 1 per supero pagina

LSR Scorrimento logico a destra

Operazione: 0→ 7 6 5 4 3 2 1 0→ C

N	Z	C	I	D	V
0	√	√	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Accumul.	LSR A	4A	1	2
Pag. zero	LSR Op	46	2	5
Pag. zero,X	LSR Op,X	56	2	6
Assoluto	LSR Op	4E	3	6
Assoluto,X	LSR Op,X	5E	3	7

NOP Nessuna operazione

Operazione: Nessuna (2 cicli)

N	Z	C	I	D	V
-	-	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	NOP	EA	1	2

ORA OR logico fra l'accumulatore e la memoria

Operazione: AVM→A

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Immediato	ORA #Op	09	2	2
Pag. zero	ORA Op	05	2	3
Pag. zero,X	ORA Op,X	15	2	4
Assoluto	ORA Op	0D	3	4
Assoluto,X	ORA Op,X	1D	3	4*
Assoluto,Y	ORA Op,Y	19	3	4*
(Ind.,X)	ORA (Op,X)	01	2	6
(Ind.),Y	ORA (Op),Y	11	2	5

* Aggiungere 1 per supero pagina

PHA Pone l'accumulatore sullo stack

Operazione: A↓

N	Z	C	I	D	V
-	-	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	PHA	48	1	3

PHP Pone il registro di stato sullo stack

Operazione: P↓

N	Z	C	I	D	V
-	-	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	PHP	08	1	3

PLA Recupera l'accumulatore dallo stack

Operazione: A↑

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	PLA	68	1	4

PLP Recupera il registro di stato dallo stack

Operazione: P↑

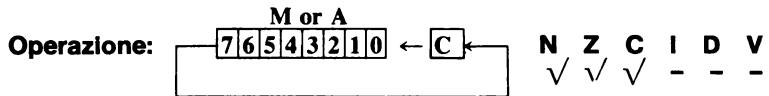
N Z C I D V
dallo stack

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	PLP	28	1	4

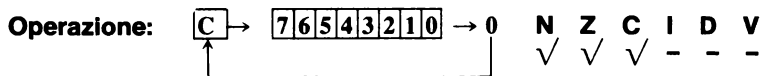
ROL Rotazione a sinistra

M or A

Operazione: 7 6 5 4 3 2 1 0 ← C

N	Z	C	I	D	V
√	√	√	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Accumul.	ROL A	2A	1	2
Pag. zero	ROL Op	26	2	5
Pag. zero,X	ROL Op,X	36	2	6
Assoluto	ROL Op	2E	3	6
Assoluto,X	ROL Op,X	3E	3	7

ROR Rotazione a destra

Operazione: C → 7 6 5 4 3 2 1 0 → 0

N	Z	C	I	D	V
√	√	√	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Accumul.	ROR A	6A	1	2
Pag. zero	ROR Op	66	2	5
Pag. zero,X	ROR Op,X	76	2	6
Assoluto	ROR Op	6E	3	6
Assoluto,X	ROR Op,X	7E	3	7

RTI Ritorno da interruzione

Operazione: P↑PC↑ **N Z C I D V**
dallo stack

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	RTI	40	1	6

RTS Ritorno da subroutine

Operazione: PC↑,PC+1→PC **N Z C I D V**
– – – – – –

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	RTS	60	1	6

SBC Sottrae la memoria e il carry dall'accumulatore

Operazione: A-M-C→A

N	Z	C	I	D	V
√	√	√	-	-	√

Indirizzam.	Assembly	Code	Byte	Cicli
Immediato	SBC #Op	E9	2	2
Pag. zero	SBC Op	E5	2	3
Pag. zero,X	SBC Op,X	F5	2	4
Assoluto	SBC Op	ED	3	4
Assoluto,X	SBC Op,X	FD	3	4*
Assoluto,Y	SBC Op,Y	F9	3	4*
(Ind.,X)	SBC (Op,X)	E1	2	6
(Ind.),Y	SBC (Op),Y	F1	2	5*

* Aggiungere 1 per supero pagina

SEC Carry a 1

Operazione: 1→C

N	Z	C	I	D	V
-	-	1	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	SEC	38	1	2

SED Decimale a 1

Operazione: 1→D

N	Z	C	I	D	V
-	-	-	-	1	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	SED	F8	1	2

SEI Interruzione a 1

Operazione: 1→I

N	Z	C	I	D	V
–	–	–	1	–	–

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	SEI	78	1	2

STA Copia l'accumulatore in memoria

Operazione: A→M

N	Z	C	I	D	V
–	–	–	–	–	–

Indirizzam.	Assembly	Code	Byte	Cicli
Pag. zero	STA Op	85	2	3
Pag. zero,X	STA Op,X	95	2	4
Assoluto	STA Op	8D	3	4
Assoluto,X	STA Op,X	9D	3	5
Assoluto,Y	STA Op,Y	99	3	5
(Ind.,X)	STA (Op,X)	81	2	6
(Ind.),Y	STA (Op),Y	91	2	6

STX Copia il registro X in memoria

Operazione: X→M

N	Z	C	I	D	V
–	–	–	–	–	–

Indirizzam.	Assembly	Code	Byte	Cicli
Pag. zero	STX Op	86	2	3
Pag. zero,Y	STX Op,X	96	2	4
Assoluto	STX Op	8E	3	4

STY Copia il registro Y in memoria

Operazione: Y→M

N	Z	C	I	D	V
-	-	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Pag. zero	STY Op	84	2	3
Pag. zero,X	STY Op,X	94	2	4
Assoluto	STY Op	8C	3	4

TAX Copia l'accumulatore in X

Operazione: A→X

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	TAX	AA	1	2

TAY Copia l'accumulatore in Y

Operazione: A→Y

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	TAY	A8	1	2

TSX Copia il puntatore allo stack in X

Operazione: S→X

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	TSX	BA	1	2

TXA Copia X nell'accumulatore

Operazione: X→A

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	TXA	8A	1	2

TXS Copia X nel puntatore allo stack

Operazione: X→S

N	Z	C	I	D	V
-	-	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	TXS	9A	1	2

TYA Copia Y nell'accumulatore

Operazione: Y→A

N	Z	C	I	D	V
√	√	-	-	-	-

Indirizzam.	Assembly	Code	Byte	Cicli
Implicito	TYA	98	1	2

MODI DI INDIRIZZAMENTO DELLE ISTRUZIONI E RELATIVI TEMPI DI ESECUZIONE (in cicli di clock)

	ACCUMULATORE	IMMEDIATO	PAGINA ZERO	PAGINA ZERO,X	PAGINA ZERO,Y	ASSOLUTO	ASSOLUTO,X	ASSOLUTO,Y	IMPLICITO	RELATIVO	(INDIRETTO,X)	(INDIRETTO),Y	ASSOLUTO INDIRETTO
ADC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
AND	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
ASL	2	.	5	6	.	6	7	.	.	.	.	.	.
BCC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BCS	.	.	.	.	.	.	.	.	.	2**	.	.	.
BEQ	.	.	.	.	.	.	.	.	.	2**	.	.	.
BIT	.	.	3	.	.	4	.	.	.	.	.	.	.
BMI	.	.	.	.	.	.	.	.	.	2**	.	.	.
BNE	.	.	.	.	.	.	.	.	.	2**	.	.	.
BPL	.	.	.	.	.	.	.	.	.	2**	.	.	.
BRK	.	.	.	.	.	.	.	.	7	.	.	.	.
BVC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BVS	.	.	.	.	.	.	.	.	.	2**	.	.	.
CLC	.	.	.	.	.	.	.	.	2	.	.	.	.
CLD	.	.	.	.	.	.	.	.	2	.	.	.	.
CLI	.	.	.	.	.	.	.	.	2	.	.	.	.
CLV	.	.	.	.	.	.	.	.	2	.	.	.	.
CMP		.	2	3	4	.	4	4*	4*	.	.	6	5*
CPX	.	2	3	.	.	4	.	.	.	.	.	.	.
CPY	.	2	3	.	.	4	.	.	.	.	.	.	.
DEC	.	.	5	6	.	6	7	.	.	.	.	.	.
DEX	.	.	.	.	.	.	.	.	2	.	.	.	.
DEY	.	.	.	.	.	.	.	.	2	.	.	.	.
EOR	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
INC	.	.	5	6	.	6	7	.	.	.	.	.	.
INX	.	.	.	.	.	.	.	.	2	.	.	.	.

	ACCUMULATORE	IMMEDIATO	PAGINA ZERO	PAGINA ZERO,X	PAGINA ZERO,Y	ASSOLUTO	ASSOLUTO,X	ASSOLUTO,Y	IMPLICITO	RELATIVO	(INDIRETTO,X)	(INDIRETTO),Y	ASSOLUTO INDIRETTO
INY	.	.	.	.	.	.	.	.	2	.	.	.	.
JMP	.	.	.	.	.	3	.	.	.	.	.	.	5
JSR	.	.	.	.	.	6	.	.	.	.	.	.	.
LDA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
LDX	.	2	3	.	4	4	.	4*	.	.	.	.	.
LDY	.	2	3	4	.	4	4*	.	.	.	.	.	.
LSR	2	.	5	6	.	6	7	.	.	.	.	.	.
NOP	.	.	.	.	.	.	.	.	2	.	.	.	.
ORA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
PHA	.	.	.	.	.	.	.	.	3	.	.	.	.
PHP	.	.	.	.	.	.	.	.	3	.	.	.	.
PLA	.	.	.	.	.	.	.	.	4	.	.	.	.
PLP	.	.	.	.	.	.	.	.	4	.	.	.	.
ROL	2	.	5	6	.	6	7	.	.	.	.	.	.
ROR	2	.	5	6	.	6	7	.	.	.	.	.	.
RTI	.	.	.	.	.	.	.	.	6	.	.	.	.
RTS	.	.	.	.	.	.	.	.	6	.	.	.	.
SBC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
SEC	.	.	.	.	.	.	.	.	2	.	.	.	.
SED	.	.	.	.	.	.	.	.	2	.	.	.	.
SEI	.	.	.	.	.	.	.	.	2	.	.	.	.
STA	.	.	3	4	.	4	5	5	.	.	6	6	.
STX	.	.	3	.	4	4	.	.	.	.	.	.	.
STY	.	.	3	4	.	4	.	.	.	.	.	.	.
TAX	.	.	.	.	.	.	.	.	2	.	.	.	.
TAY	.	.	.	.	.	.	.	.	2	.	.	.	.
TSX	.	.	.	.	.	.	.	.	2	.	.	.	.
TXA	.	.	.	.	.	.	.	.	2	.	.	.	.
TXS	.	.	.	.	.	.	.	.	2	.	.	.	.
TYA	.	.	.	.	.	.	.	.	2	.	.	.	.

* Aggiungere un ciclo per supero pagina
** Aggiungere un ciclo se viene eseguito il salto, e un ciclo ancora se si supera la pagina

Un ciclo di clock è la velocità a cui opera il processore, determinata dal numero di byte trasferiti da un componente logico interno ad un altro. L'8502 opera ad una velocità standard di 1 MHz, che equivale a 1000000 di cicli al secondo.

7

COME IMMETTERE PROGRAMMI IN LINGUAGGIO MACCHINA SUL COMMODORE 128

Ora che avete imparato i modi di indirizzamento, i tipi di istruzioni e i loro opcode dovete conoscere come realmente immettere queste istruzioni nella memoria del Commodore 128. Il C128 offre tre modi di inserimento delle istruzioni in una forma comprensibile al processore. Potete inserire delle istruzioni con:

1. Il monitor di linguaggio macchina incorporato (disponibile solo in modo C128).

2. L'istruzione **POKE**, con i valori decimali delle istruzioni (modi C128 e C64).

3. Un programma esterno chiamato assemblatore (assembler).

Ognuno di questi tre metodi comporta vantaggi e svantaggi. Per esempio, il monitor di linguaggio macchina incorporato è facile da usare e vi permette di programmare in linguaggio macchina senza ulteriori aiuti, come un assembler. Rende facile l'uso congiunto di BASIC e linguaggio macchina. Inoltre, potete salvare i programmi in linguaggio macchina col comando **SAVE** del monitor. Poichè state già lavorando su un codice oggetto, non c'è bisogno di compilare un codice sorgente come con l'assembler.

Sebbene queste siano caratteristiche potenti, il monitor non vi permette di lavorare con etichette o commenti. Esso infatti produce codice eseguibile (oggetto); quindi, nessun file sorgente. Il programma risultante contiene i veri riferimenti agli indirizzi, al posto dei riferimenti simbolici e dei commenti permessi dall'assembler. Quando osservate un programma in linguaggio macchina attraverso il monitor non avrete il lusso di commenti o variabili simboliche, così quando leggete il codice scritto da altri dovete proprio sapere cosa state osservando. D'altra parte, un file sorgente dell'assemblatore deve essere compilato nel codice oggetto, e spesso usato con un altro programma chiamato caricatore (loader). Tutto ciò richiede tre passi, mentre il linguaggio macchina del monitor è pronto per l'esecuzione non appena finite di scriverlo.

Il secondo metodo, usare l'istruzione POKE per inserire i codici decimali in memoria, è l'alternativa usata di solito quando non sono disponibili nè monitor nè assembler. È il caso in cui non disponiate di un assembler e lavoriate in modo C64, che non ha disponibile il monitor incorporato. Comunque, qualche volta è meglio usare POKE per piccole routine se l'applicazione che state scrivendo è più adatta al BASIC e necessitate della velocità del linguaggio macchina solo per una piccola parte del programma (sebbene per la maggior parte della gente questo metodo sia tedioso e richieda molto tempo). Usatelo solo se non avete alternative, poichè una volta inserito in memoria il codice, non avrete possibilità di listarlo come col monitor o l'assembler. Questo capitolo spiega l'inserimento dei programmi in linguaggio macchina coi primi due metodi sopra descritti. Il terzo metodo, l'uso di un assemblatore, richiede un pacchetto aggiuntivo simile al Sistema di Sviluppo Assembler del Commodore 64. Per avere dettagli specifici sull'uso di un assemblatore, riferitevi al manuale d'uso accluso nella confezione del programma.

IMMISSIONE DA MONITOR DELLE ISTRUZIONI DEL LINGUAGGIO MACCHINA

Incominciamo ad immettere istruzioni entrando da BASIC nel monitor con l'istruzione seguente:

MONITOR <RETURN>

Il Commodore 128 risponde con:

MONITOR

```
 PC     SR AC XR YR SP
;FB000  00 00 00 00 F8
```

Questi valori indicano il contenuto dei registri del processore dopo l'entrata nel monitor. Le abbreviazioni sono le seguenti:

PC — Contatore di Programma	Segna l'indirizzo dell'istruzione corrente
SR — Registro di Stato	Flag del registro di stato
AC — Accumulatore	Registro per le operazioni matematiche e logiche
XR — Registro indice X	Usato principalmente per modificare gli indirizzi
YR — Registro indice Y	Come il registro X
SP — Puntatore allo stack	Punta alla prima locazione libera in cima allo stack

Ora potete iniziare l'immissione delle istruzioni in linguaggio macchina. Il comando ASSEMBLE del monitor immette il codice dell'istruzione nella locazione di memoria specificata. Per inserire un'istruzione seguite il formato dell'esempio seguente:

A 01800 LDA #$00

Assicuratevi di lasciare almeno 1 spazio fra ogni campo. Ecco il significato di ogni campo:

<Assemble> <Indirizzo> <Opcode> <Operando>

La A sta per **ASSEMBLA un opcode**. La seconda parte (campo) è l'indirizzo dove verrà posto il codice dell'istruzione. Notate il numero esadecimale a 5 cifre. La cifra più a sinistra (0—F) indica la configurazione di banco del C128, esattamente come nel comando BANK.

Una volta immesso l'intero programma riferitevi all'indirizzo della prima istruzione immessa, per cominciare l'esecuzione. Usate il comando **GO** del monitor, oppure uscite dal monitor con **X (EXIT)** e date il comando **SYS.** Se usate SYS, indicate l'equivalente decimale dell'indirizzo della prima istruzione. Se volete poi ritornare al BASIC mettete **RTS** come ultima istruzione del programma. Spesso, dovrete usare una configurazione comprendente il Kernal, per ottenere il risultato voluto.

L'opcode è l'istruzione dell'8502 eseguita dal processore nel corso del programma. Andate alla Tavola del Set di Istruzioni dell'8502 per vedere quelle disponibili.

L'operando è l'indirizzo o valore su cui agisce l'opcode. Se precedete l'operando con un segno # (immediato), l'opcode lo assumerà come una costante, altrimenti come un indirizzo.

Ricordatevi di separare ogni campo con almeno 1 spazio, altrimenti il computer vi segnalerà un errore stampando un punto di domanda alla fine della riga.

Una volta listata una routine sullo schermo, il monitor permette delle scorciatoie per l'immissione delle istruzioni. Per avere il listato di un programma in linguaggio macchina, inserite il comando DISASSEMBLE come segue:

D 04000 04010 <**RETURN**>

La D sta per DISASSEMBLE. Il primo numero (04000) indica la locazione da cui partire per listare il programma, mentre il secondo quella finale.

Ora la scorciatoia. Poichè l'indirizzo dell'opcode è già sullo schermo, potete semplicemente muovere il cursore sul campo dell'opcode, digitarvi sopra la nuova istruzione, cancellare i caratteri inutili e premere **RETURN.** Il computer memorizza le istruzioni mostrando a sinistra i corrispondenti codici esadecimali. Questa è una maniera più facile e veloce per immettere routine in linguaggio macchina, piuttosto che scrivere ASSEMBLE e l'indirizzo ogni volta che inserite un'istruzione.

ESEGUIRE I VOSTRI PROGRAMMI IN LINGUAGGIO MACCHINA

Una volta scritta la vostra routine, potete eseguirla in tre modi differenti. Dal monitor, date i comandi GO o JUMP a Subroutine come segue:

```
G F1800 (JMP)
J F1800 (JSR)
```

La G sta per **GO** (vai), cioè salta all'indirizzo specificato ed esegui il programma. La J sta per **Salto a Subroutine**, ed è simile all'istruzione JSR.

La terza maniera di chiamare una routine in linguaggio macchina è uscire dal monitor col comando X. Ciò vi fa ritornare al BASIC. Poi, date il comando SYS seguito dall'indirizzo decimale, come segue:

```
BANK15
SYS6144
```

Il comando SYS é equivalente al GO F1800 visto prima. BANK 15 e F indicano la configurazione di memoria 15. Il Kernal, Il BASIC ed altri codici ROM risiedono in questa configurazione. La differenza fra SYS e GO è che l'esecuzione parte dal BASIC invece che dal monitor.

La routine in linguaggio macchina data più sotto pulisce lo schermo di testo. A partire dalla locazione 1024, viene posto il valore 32 in ogni locazione di schermo. Il codice carattere 32 é lo spazio, che cancella ogni posizione di carattere sullo schermo. Alla fine, un'istruzione RTS ritorna il controllo al BASIC. Ecco il programma BASIC principale e la subroutine di pulizia dello schermo in linguaggio macchina come apparirebbe nel monitor del linguaggio macchina.

```
10 FORI=1TO25
20 PRINT"RIEMPIO LO SCHERMO DI CARATTERI"
30 NEXT
40 PRINT:PRINT
50 PRINT"ORA CHIAMO LA ROUTINE IN LINGUAGGIO"
60 PRINT"ROUTINE PER PULIRE LO SCHERMO"
70 SLEEP5
80 SYSDEC("1800")
90 PRINT"LO SCHERMO È PULITO!"
```

```
MONITOR

.01800 A2 00       LDX #$00
.01802 A9 20       LDA #$20
.01804 9D 00 04    STA $0400,X
.01807 9D 00 05    STA $0500,X
.0180A 9D 00 06    STA $0600,X
.0180D 9D E7 06    STA $06E7,X
.01810 E8          INX
.01811 D0 F1       BNE $1804
.01813 60          RTS
```

In questo programma di esempio, il comando SYS esegue la subroutine che pulisce lo schermo di testo. Una volta fatto, il controllo viene ritornato al BASIC tramite RTS, e viene stampato READY.

COMANDI DEL MONITOR DI LINGUAGGIO MACCHINA

Il monitor incorporato del C128 possiede parecchi comandi per intervenire sulle vostre routine in linguaggio macchina, una volta introdotte in memoria. La Figura 7–1 è un sommario di tutti i comandi disponibili nel MONITOR.

PAROLA CHIAVE	FUNZIONE		FORMATO
ASSEMBLA	Assembla una linea di codice 8502	A	<ind iniz> <opcode> [operando]
CONFRONTA	Confronta due sezioni di memoria riportandone le differenze	C	<ind iniz> <ind fin> < nuovo ind iniz>
DISASSEMBLA	Disassembla una o più linee di codice 8502	D	[<ind iniz> <ind fin>]
RIEMPI	Riempie una zona di memoria col byte specificato	F	<ind iniz> <ind fin> <byte>
SALTA	Inizia l'esecuzione dall'indirizzo specificato	G	[indirizzo]
CERCA	Ricerca in memoria uno o più byte specificati	H	<ind iniz> <ind fin> <byte1> [<byte n>...]
		H	<ind iniz> <ind fin> <stringa di caratteri>
SALTA E RITORNA	Esegue un JSR ad una subroutine	J	[indirizzo]
CARICA	Carica un file da nastro o disco	L	"<nome del file>"[,<unità> [,<ind caric>]]
MEMORIA	Mostra il contenuto esadecimale della memoria	M	[<ind iniz>[<ind fin>]]
REGISTRI	Mostra il contenuto dei registri dell'8502	R	
SALVA	Salva un file su nastro o disco	S	"<nome del file>",<unità>,<ind iniz><ind fin+1>
TRASFERISCI	Trasferisce una zona da un indirizzo ad un altro	T	<ind iniz><ind fin><nuovo ind iniz>
VERIFICA	Verifica un file su nastro o disco	V	"<nome del file>"[,<unità> [,<ind caric>]]
ESCI	Esce dal monitor tornando al BASIC	X	
punto	Assembla una linea di codice 8502	.	
maggiore di	Modifica la memoria	>	[indirizzo]
punto e virgola	Modifica i registri dell'8502	;	
chiocciola	Funzioni del disco	@	
			@ [unità]
	mostra lo stato del disco		@ [unità] <stringa di comando>
	cambia il disco corrente		@ [unità],$[<drive>:<tipo di file>]

NOTA: <> racchiudono i parametri necessari
[]racchiudono i parametri opzionali

Figura 7-1. Sommario dei comandi del monitor del C128

> **NOTA:** indirizzi a 5 cifre
> Nel monitor il Commodore 128 mostra degli indirizzi a 5 cifre. Di solito, un numero esadecimale è composto da sole 4 cifre, che rappresentano il campo indirizzabile. La cifra più a sinistra indica la configurazione di banco (per il comando del momento) secondo la seguente tavola di configurazione:

0 – solo RAM 0
1 – solo RAM 1
2 – solo RAM 2
3 – solo RAM 3
4 – ROM interna, RAM 0, I/O
5 – ROM interna, RAM 1, I/O
6 – ROM interna, RAM 2, I/O
7 – ROM interna, RAM 3, I/O
8 – ROM esterna, RAM 0, I/O
9 – ROM esterna, RAM 1, I/O
A – ROM esterna, RAM 2, I/O
B – ROM esterna, RAM 3, I/O
C – KERNAL + interna bassa, RAM 0, I/O
D – KERNAL + esterna bassa, RAM 1, I/O
E – KERNAL + BASIC, RAM 0, ROM CARATT.
F – KERNAL + BASIC, RAM 0, I/O

SOMMARIO DEI DESCRITTORI DI CAMPO DEL MONITOR

I descrittori seguenti precedono i campi dei dati (per esempio, visione della memoria). Quando il monitor li incontra da soli, esso cambierà i contenuti della memoria o dei registri con i dati specificati.

. <punto>	precede le linee di disassemblato
> <maggiore>	precede le linee dei contenuti della memoria
; <punto e virgola>	precede le linee dei contenuti dei registri

I descrittori seguenti precedono i campi numerici (per esempio, un indirizzo) e specificano la base numerica del valore. Come comandi, dicono al monitor di mostrare il valore dato in tutte le quattro basi disponibili.

<spazio>	(default) precede i valori esadecimali
$ <dollaro>	precede anch'esso i valori esadecimali
+ <più>	precede i valori decimali
& <e romana>	precede i valori ottali
% <percento>	precede i valori binari

I caratteri seguenti sono usati dal monitor come delimitatori di campo o terminatori di linea (a meno che non si trovino in una stringa).

<spazio>
delimitatore – separa due campi
, <virgola>
delimitatore – separa due campi
: <due punti>
terminatore – fine della logica
? <punto int>
terminatore – fine della linea logica

DESCRIZIONE DEI COMANDI DEL MONITOR

Qui di seguito vi sono le descrizioni di ogni singolo comando del monitor.

COMANDO: **A**
SCOPO: Immette una linea di codice
SINTASSI: **A** <indirizzo><opcode><operando>

<indirizzo> Un numero a 5 cifre indicante la locazione dove porre il codice (guardate la nota sugli indirizzi a 5 cifre, nella pagina precedente).
<opcode> Un mnemonico standard dell'8502
<operando> L'operando, quando richiesto, può essere sia un indirizzo che una costante

Premete <**RETURN**> per indicare la fine della linea. Se c'è un errore verrà stampato un punto interrogativo per indicarlo. Potrete poi usare l'editor di schermo per correggere la linea.

ESEMPIO:

```
.A 01200 LDX #$00
.A 01202
```

NOTA: un punto (.) equivale al comando A

ESEMPIO:

```
.02000 LDA #$23
```

COMANDO: **C**
SCOPO: Confronta due aree di memoria
SINTASSI: **C** <ind iniz><ind fin><ind iniz>
<ind iniz> Indirizzo di partenza della prima zona
<ind fin> Indirizzo di fine della prima zona
<ind iniz> Indirizzo di partenza della seconda zona
Gli indirizzi i cui contenuti differiscono vengono stampati sullo schermo

COMANDO: **D**
SCOPO: Disassembla una linea di codice
SINTASSI: **D** [<ind iniz>][<ind fin>]
D <ind iniz> Indirizzo d'inizio del disassemblato
D <ind fin> Indirizzo di fine (opzionale)

Il formato del disassemblato differisce leggermente da quello dell'assemblato. Il primo carattere della linea è un punto invece che una A, e vengono stampati anche i codici esadecimali.

Un disassemblato può essere modificato semplicemente modificando una linea e premendo <RETURN>

Se scrivete solo D **RETURN** senza indirizzi, il disassemblato continuerà una pagina alla volta.

ESEMPIO:

```
D3000 3003
.03000 A9 00 LDA #$00
.03002 FF     ???
.03003 D0 2B BNE $3030
```

COMANDO: **F**
SCOPO: Riempie delle locazioni con un byte specificato
SINTASSI: **F** <ind iniz><ind fin><byte>
<ind iniz> Locazione d'inizio
<ind fin> Locazione finale
<byte> Valore con cui riempire le locazioni

Questo comando è utile per inizializzare delle strutture di dati o qualunque area di RAM.

ESEMPIO:

F0400 0518 EA
Riempie le locazioni da $0400 a $0518 col byte $EA (istruzione NOP).

COMANDO: **G**
SCOPO: Salta all'indirizzo specificato ed esegue il programma
SINTASSI: **G** [<indirizzo>]
<indirizzo> Un indirizzo da cui partire. Se omesso, l'esecuzione comincerà dal valore del PC (immesso col comando R)

Il comando **G** ripristina tutti i registri come mostrati dall'istruzione R. Per tornare al monitor il programma deve finire con un'istruzione BRK.

ESEMPIO:

G 140C
Inizia l'esecuzione da $140C nel banco 0. Alcune applicazioni possono richiedere la presenza del Kernal o dell'I/O: precedete quindi l'indirizzo a 4 cifre con la cifra di configurazione.

COMANDO: **H**
SCOPO: Ricerca uno o più byte od una stringa nella zona delimitata dagli indirizzi specificati
SINTASSI: **H** <ind iniz><ind fin><dati>
<ind iniz> Indirizzo d'inizio della ricerca
<ind fin> Indirizzo di fine
<dati> Numeri o lettere da cercare

ESEMPIO:

H A000 A101 A9
Cerca il valore $A9 da $A000 a $A101.
H 2000 9800 'CASSA'
Cerca la stringa alfabetica CASSA.

COMANDO: **J**
SCOPO: Salta e ritorna da una subroutine
SINTASSI: **J** <indirizzo>

Questo comando esegue una subroutine, salvando l'indirizzo di ritorno esattamente come JSR. Quando, nella subroutine, si incontrerà RTS, il controllo ritornerà al monitor.

ESEMPIO:

J 2000
Esegue un JSR alla subroutine in $2000 nel banco 0.

COMANDO: **L**
SCOPO: Carica un file da nastro o disco
SINTASSI: **L** <"nome del file">[,<unità>[,ind caric]]

<"nome del file">	Ogni nome valido
<unità>	Numero del dispositivo (1:cass.; 8,9: disco)
[ind caric]	Un indirizzo di caricamento (opzionale).

Il comando CARICA fa sì che un file sia caricato in memoria. L'indirizzo di caricamento viene ricavato dalla testata del file su nastro o dai primi due byte del file su disco. In altre parole il comando CARICA carica sempre un file nelle stesse locazioni da cui era stato salvato. Questo é particolarmente importante lavorando in linguaggio macchina, poichè è raro che un programma sia rilocabile. Il caricamento continuerà fino all'**EOF** (End Of File: fine del file).

ESEMPIO:

L"PROGRAMMA",8 Carica il file di nome PROGRAMMA da disco.

COMANDO: **M**
SCOPO: Mostra il contenuto esadecimale ed ASCII degli indirizzi specificati
SINTASSI: **M** [<ind iniz>][<ind fin>]

<ind iniz>	Inizio della zona da visualizzare. La prima cifra é la configurazione di banco, le 4 seguenti l'indirizzo.
<ind fin>	Fine della zona. Pure questo usa 5 cifre.

Se omettete uno o entrambi gli indirizzi, verrà stampata la pagina corrente.

La memoria viene visualizzata nel formato seguente:

> 1A048 41 42 43 44 45 46 47 48:ABCDEFGH

I contenuti della memoria possono essere cambiati semplicemente digitando i nuovi valori e premendo <**RETURN**>. Nel caso ci sia un errore o si tenti di scrivere su una ROM, verrà stampato un punto di domanda. I dati vengono visualizzati in esadecimale e, a destra, come caratteri in negativo (per differenziarli). Nel caso non siano stampabili, verrà messo un punto. Pure questo comando, dato da solo, mostra la memoria a pagine.

ESEMPIO:

M 21C00
> 21C00 41 4A 4B 4C 4D 4E 4F 50 :AJKLMNOP

> **NOTA:** Lo schermo a 40 colonne visualizzerà 8 valori, mentre quello a 80 il doppio, cioè 16.

COMANDO: **R**

SCOPO: Mostra i registri principali dell'8502 e permette di modificarli.
Verranno quindi stampati il registro di stato, il contatore di programma, l'accumulatore, i registri X ed Y ed il puntatore allo stack. I dati visualizzati verranno copiati nei registri reali del processore all'esecuzione di un comando G o J.

SINTASSI: **R**

ESEMPIO:

```
R
PC      SRACXRYRSP
;01002  01 02 03 04 F6
```

> **NOTA:** il punto e virgola (;) può essere usato per modificare i registri come il > del comando M.

COMANDO: **S**

SCOPO: Salva i contenuti della memoria su nastro o disco

SINTASSI: **S** <"nome del file">,<unità>,<ind iniz>,<ind fin+1>

<nome del file>	Ogni nome valido, racchiuso fra virgolette (non apici)
<unità>	Il numero del dispositivo (1: cass; 8, 9: disco)
<ind iniz>	Inizio della zona da salvare
<ind fin+1>	Indirizzo finale + 1 della zona da salvare. Il byte in <ind fin+1> non verrà salvato.

Il comando creerà un file programma, i cui primi due byte saranno <ind iniz>. Per ricaricarlo usate il comando L.

ESEMPIO:

S"GIOCO",8,0400,0C00
Salva su disco la memoria da $0400 a $0C00 (escluso).
La zona salvata va da $0400a $0BFF.

COMANDO: **T**
SCOPO: Trasferisce una zona di memoria
SINTASSI: **T** <ind iniz><ind fin><ind iniz>
<ind iniz> Inizio della zona da trasferire
<ind fin> Fine della zona da trasferire (compreso)
<ind iniz> Inizio della zona in cui trasferire

I dati possono essere trasferiti dalla parte alta della memoria a quella bassa e viceversa. Persino segmenti inizianti in un banco e finenti in un altro (ad esempio dal banco 0 al banco 1) possono essere manipolati. Durante il trasferimento viene inoltre eseguito un confronto automatico, indicando gli indirizzi che hanno registrato un byte diverso da quello trasferito (ad esempio zone vuote senza RAM).

ESEMPI:

T 1400 1600 1401
Sposta i dati da $1400 a $1600 (compreso) in alto di 1 byte, da $1401.

T F000 10800 2000
Sposta la zona di memoria costituita da $F000—FFFF nel banco 0 e $10000—10800 nel banco 1, partendo da $2000 nel banco 0. Si possono così considerare tutti i 128K come un unico banco.

COMANDO: **V**
SCOPO: Verifica un file su nastro o disco
SINTASSI: **V** <"nome del file">[,<unità>][,ind verifica]
<"nome del file"> Ogni nome valido
<unità> Numero di dispositivo (1: cass; 8, 9: disco)
[ind verifica] Indirizzo (opzionale) da cui verificare

Il comando V confronta il file in memoria con quello su disco: se c'è un errore verrà stampato VERIFY ERROR, altrimenti nessun messaggio.

ESEMPIO:

V"WORKLOAD",8

COMANDO: **X**
SCOPO: Esce dal monitor
SINTASSI: **X**

COMANDO: > (maggiore)
SCOPO: Usato per modificare fino a 8 (16 in 80 colonne) locazioni di memoria
SINTASSI: > <indirizzo> <byte1> [<byte n>...]
<indirizzo> Indirizzo della zona di memoria
<byte1> Byte da porre all'indirizzo specificato
<byte n>] Successivi byte (opzionali), separati da almeno 1 spazio, e da porre agli indirizzi seguenti.

COMANDO: @ (chiocciola)
SCOPO: Operazioni col drive
SINTASSI: @ [<unità>],stringa di comando
<unità> Un drive da 8 a 11
<stringa di comando>Comando da inviare al drive

NOTA: @ da solo legge lo stato del drive.

ESEMPI:

@	Legge lo stato del drive 00,OK,00,00
@,I	Inizializza il drive 8
@,$	Stampa la Directory dell'unità 8
@,$0:F*	Stampa tutti i file che iniziano per F sul drive 0, unità 8

Come ulteriore aiuto al programmatore, mentre si lavora nel monitor è abilitata la capacità del Kernal di stampare gli errori per numero e non per stringa, così da scrivere I/O ERROR # e il numero dell'errore. Ciò viene disabilitato all'uscita dal monitor.

MANIPOLAZIONE DEL TESTO TRAMITE IL MONITOR DI LINGUAGGIO MACCHINA

Alcuni programmi applicativi in linguaggio macchina richiedono la manipolazione di stringhe di caratteri. Se usate un assembler, esso vi permetterà di inserire delle stringhe nel sorgente. Comunque, nel monitor, le stringhe devono risiedere in memoria, sia modificando la memoria tramite l'editor, sia inserendo i caratteri da programma.

Per modificare la memoria con l'editor di schermo date il comando **M** seguito dal campo d'indirizzo relativo alla zona dove porre le stringhe. Per esempio, supponiamo vogliate mettere la parola "TESTO" partendo dalla locazione $2000. Primo, entrate nel monitor col comando MONITOR. Poi, date il comando seguente:

```
M 2000
```

Il 128 risponde con:

```
02000 FF 00 FF 00 FF 00 FF 00:π .π .π .π .
```

L'intero schermo viene riempito con il contenuto delle locazioni di memoria da $2000 a $205F. A scopo dimostrativo, qui viene stampata solo una linea. Questa linea è relativa agli indirizzi da $2000 a $2007. Sull'estrema destra dello schermo c'è un'area che mostra i caratteri corrispondenti ai codici esadecimali. Il carattere più a sinistra si riferisce alla locazione $2000, quello immediatamente a destra alla $2001, e così via. Per porre in memoria la parola "TESTO" dalla locazione $2000, muovete il cursore sul primo codice esadecimale subito a destra dell'indirizzo, e digitate il codice della lettera T (84 decimale, $54 esa), poi premete <**RETURN**>. A questo punto potete vedere la lettera T in negativo sulla destra dello schermo. Riferitevi all'Appendice E, Volume 8, Codici ASCII e CHR$, per un elenco di tutti i caratteri disponibili sul Commodore 128.

Ora seguite la medesima procedura per le lettere E, S, T e O. Una volta fatto, vedrete la parola "TESTO" nell'area a destra. La prima linea apparirà dunque così:

```
02000 54 45 53 54 4F 00 FF 00:TESTO .π . π
```

Ora la stringa di caratteri di cui avete bisogno è in memoria, dall'indirizzo $2000 in poi. La vostra routine in linguaggio macchina può ora agire sui caratteri della parola "TESTO" così da stamparla sullo schermo. Un modo efficiente di manipolare intere parole consiste nell'usare l'indirizzo d'inizio del testo, in questo caso $2000. Determinate la lunghezza della stringa, ed usate un registro indice come uno spostamento fino alla fine della stringa. Guardate nel Capitolo 9, Programma a schermo diviso tramite raster interrupt con scroll orizzontale, per avere un esempio funzionante di manipolazione del testo. Questo capitolo ha descritto l'uso del linguaggio macchina. Per ulteriori informazioni sul linguaggio macchina, andate ai Capitoli 6, 8, 9, 10, 12, e 14.

8

USO COMBINATO DI LINGUAGGIO MACCHINA E BASIC

PERCHÈ USARE IL BASIC ASSIEME AL LINGUAGGIO MACCHINA?

Certi programmi di applicazione sono più adatti ad un linguaggio ad alto livello come il BASIC piuttosto che al linguaggio macchina, a basso livello. In altri casi, comunque, certe parti di programma, come quelle relative alla grafica, possono richiedere la velocità del linguaggio macchina mentre il resto del programma si presta meglio al BASIC. Questa è la ragione principale per usare programmi misti di BASIC e linguaggio macchina. Un'altra ragione può essere la mancanza di alternative di programmazione in linguaggio macchina. Per esempio, in modo C64 non è normalmente disponibile un monitor. Inoltre, potreste anche non avere un pacchetto assembler, così l'unica alternativa resta l'introduzione dei programmi in linguaggio macchina attraverso il BASIC. Questo metodo ha degli svantaggi; può essere lungo e tedioso, e una volta introdotta la routine in memoria, non avrete modo di listarla per correggerla. Questo metodo è raccomandato solo come ultima alternativa.

MEMORIZZAZIONE DA BASIC DELLE ROUTINE IN LINGUAGGIO MACCHINA

Nel Capitolo 7, avete visto un esempio dell'uso del comando SYS per passare dal BASIC ad una routine in linguaggio macchina per pulire lo schermo. Il comando SYS chiamò la routine, pulì lo schermo e tramite RTS tornò al BASIC. Questo esempio mostra il comando SYS dato da programma. Potete chiamare una routine anche da fuori un programma, come adesso:

```
SYS 8192
```

Questo esempio presume abbiate introdotto la routine tramite il monitor (e che essa sia ancora in memoria). Cosa potete fare se non avete un monitor, come in modo 64, e volete aggiungere una routine in linguaggio macchina al vostro programma BASIC?

La risposta a questo consiste nell'usare l'istruzione POKE per inserire in memoria i dati decimali relativi alle istruzioni e agli operandi. Poi, per attivarla, date SYS come visto prima. Questo metodo richiede i seguenti passi:

1. Scrivete il vostro programma su un pezzo di carta

2. Traducete i codici mnemonici nei corrispondenti codici decimali. Alcune istruzioni richiederanno 3 byte, altre 1 o 2. Per avere un elenco delle istruzioni e dei codici corrispondenti andate alla Tabella delle Istruzioni e dei Modi d'Indirizzamento dell'8502, nel Capitolo 6.

3. Trascrivete i codici decimali in istruzioni DATA, in modo da avere un'istruzione per linea, così:

```
1000 DATA 162,0:REM = LDX #$00 = $A2,$00
```

Il numero esadecimale $A2 rappresenta l'istruzione LDX dell'8502,equivalente a 162 decimale. Lo 0 rappresenta l'operando 0, che viene caricato dall'istruzione nel registro X. In essa, 0 è identico al decimale, così il secondo byte è sempre 0. Gli opcode esadecimali sono poi tradotti in stringhe di bit così che il processore li possa interpretare ed eseguire. Questo è il vero linguaggio macchina al livello più basso.

4. Una volta tradotti in decimale tutti i codici, dovete introdurli in memoria. Per fare ciò usate READ da BASIC e poi POKE per inserirli in memoria all'indirizzo appropriato. Per esempio, per inserire l'istruzione LDX #$00, in modo C128, eseguite la seguente routine:

```
      10 ALFA=8192
      20 I=0
      30 DO
      40 READ A
      45 IF A=-999THEN EXIT
      50 POKE ALFA+I,A
      60 I=I+1
      70 LOOP
      80 PRINT"TUTTI I DATI SONO ORA IN MEMORIA"
      :
      :
1000 DATA162,0,-999
```

Per il modo C64 usate questa routine:

```
10 ALFA=8192
20 FOR I=0TO1
40 READ A
50 POKE ALFA+I,A
60 NEXT
80 PRINT"TUTTI I DATI SONO ORA IN MEMORIA"
:
:
1000 DATA162,0
```

Il primo esempio, in Modo C128, legge tutti i dati e li inserisce in memoria partendo dall'indirizzo 8192 ($2000), fino alla lettura di −999. Quando viene letto −999, l'istruzione EXIT fa uscire dal ciclo e viene stampato un messaggio che avverte che tutti i dati sono in memoria. Questo **DO...LOOP** vi permette di immettere quanti dati vogliate senza bisogno di conoscerne la quantità, poichè la fine della lista è segnalata da −999. Questo approccio permette di modificare facilmente il numero dei dati senza modificare alcunchè del programma. In questo programma si dà per scontato che la bit map sia disallocata. ($2000−$3FFF).
Il secondo esempio, in modo C64, usa un ciclo **FOR...NEXT** per memorizzare i dati. Ciò vi richiede di conoscere l'esatto numero dei dati nella lista. Potete aggiungere un'istruzione **IF...THEN** per controllare un valore finale come −999, come nel primo esempio. Comunque, questo metodo illustra un modo diverso di eseguire la stessa cosa.

5. Il passo finale per eseguire routine in linguaggio macchina da BASIC è eseguire la subroutine col comando SYS. Aggiungete alle vostre routine BASIC la linea 90, come segue:

```
90 SYS 8192
```

Questo comando fa partire la routine in linguaggio macchina che avete inserito tramite POKE partendo dalla locazione 8192.

Sebbene l'istruzione DATA dell'esempio nel Passo 4 non lo mostri, tutte le subroutine in linguaggio macchina devono terminare con un'istruzione RTS così da tornare al BASIC. Il codice decimale per RTS è 96. Il vostro ultimo dato decimale nell'istruzione DATA finale deve essere 96, a meno che non usiate un terminatore come −999; in quel caso −999 sarà l'ultimo elemento.

La Figura 8-1 mostra una traduzione passo-passo della routine in linguaggio macchina che pulisce lo schermo da come appare nel monitor, e un programma completo che mischia la routine ad un programma BASIC che la inserisce in memoria e la esegue. Funziona solo sullo schermo a 40 colonne (VIC).

Indirizzo	Codice esa	Istruzione simbolica		Equivalente decimale		
				1°byte (opcode)	2°byte (operando)	3°byte
. 02000	A2 00	LDX #$00	=	162	0	
. 02002	A9 20	LDA #$20	=	169	32	
. 02004	9D 00 04	STA $0400,X	=	157	0	4
. 02007	9D 00 05	STA $0500,X	=	157	0	5
. 0200A	9D 00 06	STA $0600,X	=	157	0	6
. 0200D	9D E7 06	STA $06E7,X	=	157	231	6
. 02010	E8	INX	=	232		
. 02011	D0 F1	BNE $2004	=	208	241	
. 02013	60	RTS	=	96		

Figura 8-1. Traduzione passo-passo in decimale

Per trovare i codici esadecimali, riferitevi alla Tabella delle Istruzioni e dei Modi d'Indirizzamento dell'8502 nel Capitolo 6. Notate, nella Figura 8-1, che i codici esadecimali sono stampati a sinistra dei mnemonici. Questi codici esa sono i decimali che trascriverete nelle istruzioni DATA.

Notate il secondo byte, 241, nell'istruzione BNE. In un'istruzione di salto relativo, l'operando non è un indirizzo assoluto ma uno spostamento verso un'altra istruzione. In questo caso, l'istruzione BNE salta indietro alla locazione $2004; quindi, salta indietro di 15 locazioni fino all'istruzione **STA**.

Vi starete probabilmente domandando come fa il codice 241 a dire al computer di saltare indietro di 15. Il numero 241 è il complemento a 2 di 15. Quando il bit 7 vale 1, il processore salta all'indietro. 241 significa quindi di saltare indietro di 15 locazioni e ricominciare l'esecuzione da là. Per trovare il valore originario di un numero in complemento a 2 fate così:

1. Traducete il numero in binario: 241 = 11110001
2. Sottraete 1: = 11110000 = 240
3. Eseguite l'OR esclusivo del risultato: = 00001111 = 15

Per trovare il complemento a 2 di un numero, eseguite i passi 1 e 3, poi aggiungete 1.

Ecco il programma completo in modo C128, compresi tutti i dati decimali equivalenti alle istruzioni in linguaggio macchina della routine di pulizia dello schermo:

```
10 ALFA=8192
20 I=0
30 DO
40 :        READ A
45 :        IF A=-999THEN EXIT
50 :        POKE ALFA+I,A
60 :        I=I+1
70 LOOP
80 PRINT"TUTTI I DATI SONO ORA IN MEMORIA
85 SLEEP1
90 SYS8192
1000  DATA162,0,169,32,157,0,4,157,0,5,157,0,6,157,231,6
2000  DATA232,208,241,96,-999
```

Ecco il programma corrispondente in modo C64:

```
10 ALFA=8192
20 FOR I=0 TO 19
40 :        READ A
50 :        POKE ALFA+I,A
60 NEXT
80 PRINT"TUTTI I DATI SONO ORA IN MEMORIA"
85 FOR I=1 TO 2500:NEXT
90 SYS8192
1000  DATA162,0,169,32,157,0,4,157,0,5,157,0,6,157,231,6
2000  DATA232,208,241,96
```

Quando eseguirete questo programma, il computer leggerà le istruzioni DATA tramite READ, memorizzerà i dati con POKE ed eseguirà la routine di pulizia col comando SYS. Se siete in modo C128, una volta eseguito il programma entrate in monitor e disassemblate il codice da $2000 a $2015 col comando:

```
D 2000 2015
```

Notate che la subroutine inserita con POKE è la stessa della Figura 8-1. I due differenti metodi portano allo stesso risultato — programmare in linguaggio macchina dell'8502.

DOVE PIAZZARE IN MEMORIA I PROGRAMMI IN LINGUAGGIO MACCHINA

Il Commodore 128 ha 128K di memoria RAM, divisi in 2 banchi da 64K. Molti dei 128K di RAM sono ricoperti dalla ROM, ma non contemporaneamente. La memoria del C128 è a strati, così la RAM è al di sotto della ROM che la ricopre. I progettisti del Commodore 128 hanno fatto in modo di comprimere 28K di ROM e 128K di RAM in 128K di spazio d'indirizzamento. Solo un banco è disponibile ad un dato momento, poichè il più alto indirizzo di un processore a 8 bit è 65535 ($FFFF). Comunque, dato che il C128 è capace di alternare tra RAM e ROM così velocemente, potrebbe sembrare come se ci fossero tutti i 128K disponibili.

Nella parte di memoria condivisa da RAM e ROM, un'operazione di lettura ritornerà il valore della ROM, mentre una di scrittura andrà sulla RAM sotto la ROM che la ricopre. Se i dati nella RAM sotto la ROM sono un programma, la ROM sopra dovrà essere disabilitata prima dell'esecuzione del programma. La disposizione di RAM e ROM in memoria è controllata attraverso il **Registro di Configurazione (CR)** dell'**Unità di Gestione della Memoria (MMU)**. Per informazioni più dettagliate, riferitevi alla sezione sui Registri dell'Unità di Gestione della Memoria (in specie, la parte sul Registro di Configurazione) nel Capitolo 14.

DOVE PIAZZARE LE ROUTINE IN LINGUAGGIO MACCHINA CON RIGUARDO AL BASIC

Da BASIC, il sistema operativo si prende cura di alternare fra RAM e ROM secondo necessità. Il sistema operativo del C128 fornisce 16 configurazioni di memoria standard. Ognuna di esse pone un valore differente nel Registro di Configurazione; quindi, ognuna ha una configurazione differente. Ciò vi permette di scegliere fra 16 diverse configurazioni di memoria. La Figura 8-2 elenca le 16 configurazioni disponibili da BASIC.

BANCO	CONFIGURAZIONE
0	solo RAM 0
1	solo RAM 1
2	solo RAM 2
3	solo RAM 3
4	ROM interna, RAM 0, I/O
5	ROM interna, RAM 1, I/O
6	ROM interna, RAM 2, I/O
7	ROM interna, RAM 3, I/O
8	ROM esterna, RAM 0, I/O
9	ROM esterna, RAM 1, I/O
10	ROM esterna, RAM 2, I/O
11	ROM esterna, RAM 3, I/O
12	KERNAL + interna bassa, RAM 0, I/O
13	KERNAL + esterna bassa, RAM 1, I/O
14	KERNAL + BASIC, RAM 0, ROM CARATT.
15	KERNAL + BASIC, RAM 0, I/O

Figura 8-2. Tavola delle Configurazioni di Banco

Se volete memorizzare una routine in linguaggio macchina col BASIC attivo, mettete la subroutine in un banco con della RAM, preferibilmente nel banco 0, poichè è composto solo da RAM. Se invece la subroutine viene posta in un banco diverso dallo 0, non tutta la RAM sarà disponibile, poichè una parte sarà ricoperta dalla ROM. Dovete controllare il valore del Registro di Configurazione in quel banco per vedere quali indirizzi contengano della ROM. Guardate il valore del Registro di Configurazione in ognuna delle 16 configurazioni e confrontatelo con la tabella di Figura 14-5 per vedere le zone di ROM.

Seguite questa procedura per chiamare da BASIC delle subroutine in linguaggio macchina:

1. Memorizzate la subroutine, preferibilmente nel banco 0, sia attraverso il monitor che da BASIC. Se il vostro programma richiede il Kernal, il BASIC e l'I/O scegliete il banco 15. Se immettete la subroutine da monitor, ponete uno 0 davanti all'indirizzo per porla nel banco 0. Se invece usate POKE da BASIC (non è il metodo che preferiamo) date il comando **BANK 0** prima di memorizzare i dati, assumendo che poniate i dati nel banco 0. Il campo ideale per le routine in linguaggio macchina va da $1300 a $1BFF. È disponibile anche l'area superiore (testo BASIC), ammesso che il vostro programma BASIC lasci dello spazio a disposizione.

2. Ora, per eseguire il resto del programma BASIC, tornate al banco 15 (il banco di default) col comando:

   BANK 15

3. Ora chiamate la subroutine tramite SYS. L'indirizzo è quello della prima istruzione del vostro programma in linguaggio macchina. In questo caso, assumiamo che la subroutine parta da $2000 (non usando lo schermo bit map) e facciamo:

   SYS 8192

La RAM della configurazione 0 nella Figura 8-2 è la stessa delle configurazioni (banchi) 4, 8, 12, 13, 14 e 15. Per esempio, potete memorizzare dei programmi nel banco 15, ma dovete assicurarvi che la RAM non sia ricoperta dalla ROM.

NOTA: Se pensate di tornare al BASIC, assicuratevi di terminare la vostra routine con un'istruzione RTS.

DOVE PIAZZARE LE ROUTINE IN LINGUAGGIO MACCHINA SENZA IL BASIC

Quando programmate in linguaggio macchina e non avete bisogno della ROM BASIC, potete escluderla dalla mappa di memoria, facendo apparire la RAM sottostante. Per fare ciò, ponete un certo valore nel Registro di Configurazione, come segue:

```
LDA #$0E     ;Valore di configurazione
STA $D501    ;Registro di Preconfigurazione
STA $FF01    ;Registro di Configurazione
```

Potete usare questa sequenza:

```
LDA #$0E
STA $FF00
```

Quando escludete il BASIC, non potrete più usare il comando BANK per selezionare le 16 configurazioni, così sarete pienamente responsabili della gestione del Registro di Configurazione. La Figura 14-5, Volume 5, definisce il significato di ogni bit del Registro di Configurazione per stabilire una data mappa.

Quando escludete il BASIC, gli indirizzi da $4000 a $7FFF (ROM bassa) e da $8000 a $BFFF (ROM alta) sono disponibili. Abbiate una certa cura nell'escludere il Kernal, poichè esso esegue tutte le operazioni del C128, comprese quelle routine trasparenti all'utente (cioè quelle che si danno per scontate).

In certi punti del vostro programma potreste aver bisogno di escludere temporaneamente l'I/O. Per esempio, quando volete ricopiare il set di caratteri, che si trova esattamente sotto l'I/O, da $D000 a $DFFF. In questo caso dovete passare nella configurazione 14, ricopiare il set e tornare nel banco 15.

Andate alla sezione che tratta del Registro di Configurazione, nel Capitolo 14, per una spiegazione completa della gestione di ROM e RAM sul C128.

Questo capitolo ha descritto l'uso combinato di BASIC e linguaggio macchina. Per l'uso del BASIC, andate al Volume 1, Capitoli 2 e 3; Volume 2, Capitoli 4 e 5. Per il linguaggio macchina, Volume 2, Capitolo 6; in questo volume, Capitoli 7 e 9; Volume 4, Capitoli 10, 11, 12; Volume 5, Capitolo 14.

9

LA POTENZA NASCOSTA DELLA GRAFICA DEL COMMODORE 128

LA RELAZIONE FRA BANCHI VIDEO, BANCHI RAM E CONFIGURAZIONI DI MEMORIA

Molti di voi sapranno come il Commodore 64 gestisce la memoria. Questa sezione spiega la gestione della memoria video da parte del C128, e la relazione fra banchi video e configurazioni di memoria.

GESTIONE DELLA MEMORIA A BANCHI

La gestione a banchi è il processo per cui una sezione di memoria viene indirizzata dal processore. Si dice dunque che la memoria nel banco è corrente quando essa è disponibile al processore.

Il Commodore 128 può programmare le sue configurazioni di memoria. Il BASIC e il Monitor di linguaggio macchina vi mettono a disposizione 16 configurazioni di default (chiamate banchi). Per gli scopi di questa trattazione, ci si riferisce ai banchi del BASIC semplicemente come configurazioni di memoria di default comprendenti ROM e RAM in vari indirizzi di memoria. La configurazione corrente, sia in BASIC che in linguaggio macchina, è determinata dal valore del registro di configurazione dell'Unità di Gestione della Memoria (MMU) del C128.

Ognuna delle 16 configurazioni richiede un valore differente per avere diverse combinazioni di RAM e ROM. Per esempio, la ROM caratteri è disponibile solo nel banco 14 (BANK14 in BASIC; il prefisso E negli indirizzi del monitor), poichè questa configurazione dice all'MMU di escludere l'I/O rimpiazzandolo con la ROM caratteri. Per riavere l'I/O, tornate in una configurazione che lo preveda. La figura 9-1 elenca le 16 configurazioni disponibili da BASIC e Monitor. Il Capitolo 14, Il Sistema Operativo del Commodore 128, contiene informazioni sulla programmazione dell'MMU.

I DUE BANCHI DA 64K

La memoria del Commodore 128 è composta da 2 banchi (chiamati 0 e 1), ognuno di 64K, per un totale di 128K. Il bus indirizzo dell'8502 è largo 16 bit, permettendogli di indirizzare fino a 65536 (64K) locazioni contemporaneamente. La Figura 9-2 mostra i due banchi da 64K separati.

Sebbene solo 64K possano essere indirizzati ad un tempo, l'MMU ha la capacità di far condividere fino a 16K di RAM comune fra i due banchi. La RAM comune verrà trattata nel Capitolo 14.

Il processore 8502 e il chip VIC possono accedere ognuno ad un differente banco di 64K. Il banco dell'8502 è selezionato dai bit 6 e 7 del registro di configurazione, mentre quello del VIC dai bit 6 e 7 del registro di configurazione della RAM. Pure questo argomento verrà trattato nel Capitolo 14.

BANCO	CONFIGURAZIONE
0	solo RAM 0
1	solo RAM 1
2	solo RAM 2
3	solo RAM 3
4	ROM interna, RAM 0, I/O
5	ROM interna, RAM 1, I/O
6	ROM interna, RAM 2, I/O
7	ROM interna, RAM 3, I/O
8	ROM esterna, RAM 0, I/O
9	ROM esterna, RAM 1, I/O
10	ROM esterna, RAM 2, I/O
11	ROM esterna, RAM 3, I/O
12	KERNAL + interna bassa, RAM 0, I/O
13	KERNAL + esterna bassa, RAM 1, I/O
14	KERNAL + BASIC, RAM 0, ROM CARATT.
15	KERNAL + BASIC, RAM 0, I/O

Figura 9-1. Configurazioni di memoria standard del C128

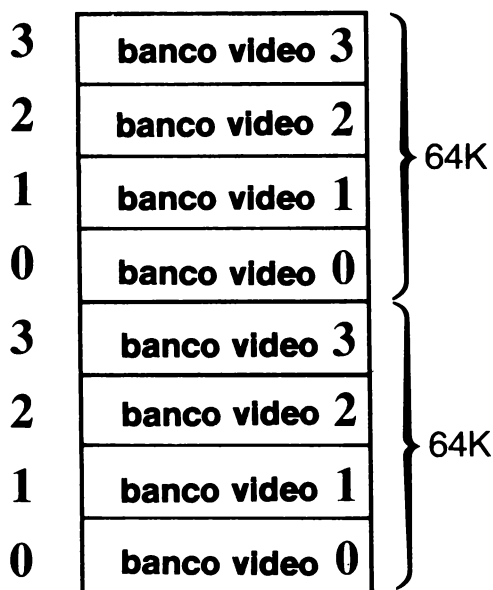



Figura 9-2. I banchi di RAM da 64K del C128

La configurazione determinata dal Registro di Configurazione può essere composta da RAM e ROM, con la ROM che ricopre la RAM sottostante, come illustrato nella Figura 9-3.

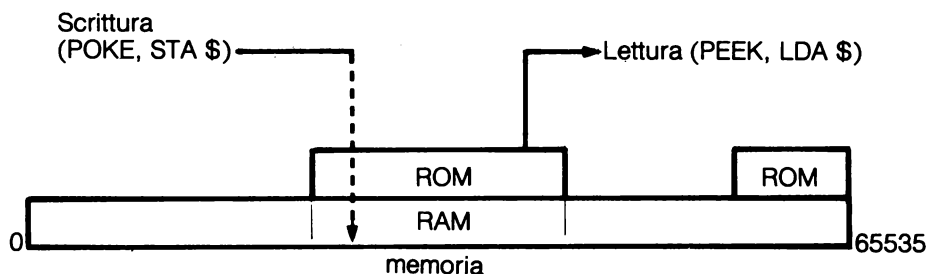



Figura 9-3. Ricopertura della ROM sulla RAM

Un'operazione di lettura ritornerà il contenuto della ROM, e una di scrittura salterà la ROM scrivendo sulla RAM sottostante.

Si possono costruire parecchie differenti configurazioni di memoria per formare uno spazio indirizzabile di 64K. I bit 6-7 del registro di configurazione specificano quale banco di RAM giace sotto gli strati di ROM specificati dai bit da 0 a 5. I banchi di RAM sono del tutto indipendenti da quelli di ROM, così potete passare dalla RAM 0 alla 1 pur mantenendo il BASIC, l'I/O e il Kernal.

I BANCHI VIDEO DA 16K

Nella programmazione grafica del C128, un banco video è un blocco di 16K di memoria che contiene le informazioni essenziali alla gestione grafica del C128: la memoria di schermo e quella dei caratteri. Questi due tipi di memoria, discussi nella sezione seguente, devono ricadere entro i 16K di memoria chiamati banco video (banco del VIC). Il chip VIC può indirizzare contemporaneamente solo 16K di memoria, così tutta la memoria grafica deve essere in quei 16K. Poichè il microprocessore del C128 indirizza fino a 64K contemporaneamente, ci sono 4 banchi video in ogni banco da 64K, per un totale di 8 banchi video (guardate la Figura 9-4).

La scelta del banco video del VIC dipende dal vostro programma applicativo. Il sistema operativo su ROM del Commodore 128 seleziona di default il banco video 0, nella parte bassa della RAM del banco 0. Le memorie di schermo e caratteri possono essere locate in posizioni differenti all'interno di ogni banco video da 16K, sebbene per programmare ottimamente il chip VIC il banco da 16K corrente debba contenerle entrambe completamente. Capirete ciò leggendo le prossime pagine.

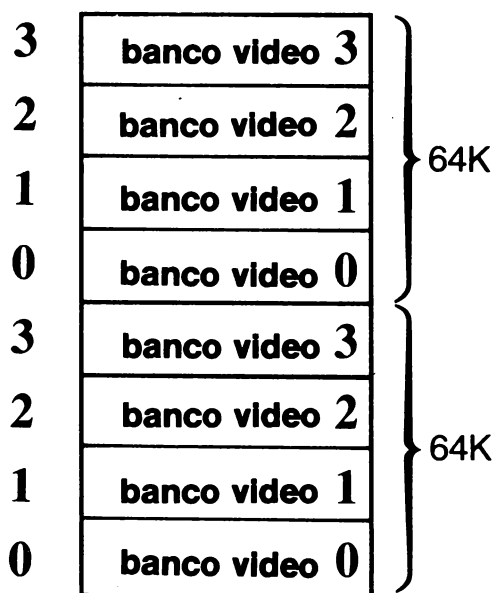



Figura 9-4. Banchi video nei banchi di RAM

I 4 banchi video in ogni banco da 64K sono allocati agli indirizzi seguenti:

BANCO	INDIRIZZI	BIT 0–1 IN $DD00	
		BINARIO	DECIMALE
0	$0000–3FFF	11=	3 (DEFAULT)
1	$4000–7FFF	10=	2
2	$8000–BFFF	01=	1
3	$C000–FFFF	00=	0

Figura 9-5. Indirizzi dei banchi video

Ogni banco di RAM (0 e 1) ha questa mappa video.
I bit 0 e 1 di $DD00 selezionano il banco video. Per scegliere da BASIC, date questo comando:

POKE 56576, (PEEK (56576) AND 252) ORX

Dove X è il valore decimale dei bit 0 e 1 della Figura 9-5.

In linguaggio macchina, usate questo programmino per selezionare i banchi:

```
LDA $DD00      ;Carica in A il contenuto di $DD00
AND #$FC       ;Azzera i bit 0 e 1
ORA #$X        ;X rappresenta il valore esa della Figura 9-5
STA $DD00      ;Ripone A in $DD00
```

Nella terza istruzione, rimpiazzate X col valore esadecimale nella Figura 9-5. Il valore di default è 3, il banco video 0.

Quando cambiate banco video, dovete aggiungere $4000 agli indirizzi delle memorie video e carattere, e bit map in modo grafico, per ogni banco sopra lo 0. Bisogna così aggiungere $4000 per il banco 1, $8000 per il 2 e $C000 per il 3. Moltiplicate dunque il numero del banco per $4000 (16384) e aggiungetelo agli indirizzi delle memorie video e carattere.

SOMMARIO DEL CONCETTO DI BANCO

Le caratteristiche maggiori del concetto di banco sono:

1. Il BASIC e il Monitor possiedono 16 configurazioni da 64K di memoria che danno 16 differenti mappe di memoria. Il chip MMU, e in specie il valore nel registro di configurazione, controlla parecchi aspetti della gestione della memoria sul C128. Per leggere o scrivere una certa parte di memoria, dovete scegliere una configurazione che la contenga. La Figura 9-1 elenca le 16 configurazioni di default disponibili da BASIC e da Monitor.

2. I 128K di memoria sono divisi in 2 banchi da 64K. Solo un banco è indirizzabile dal processore in un determinato momento. La selezione dei banchi di RAM è controllata dal registro di configurazione dell'MMU (bit 6 e 7), che fa parte dell'I/O del C128. Il chip VIC e l'8502 possono accedere entrambi ad un differente banco di 64K. La Figura 9-2 illustra i 2 banchi separati da 64K.

3. Ogni banco da 64K è diviso in 4 segmenti video da 16K. Le memorie video e carattere devono risiedere entrambe nel banco video selezionato per visualizzare correttamente testo e grafica. Per ogni banco video maggiore di 0, ricordate di aggiungere $4000 (16384) all'indirizzo di partenza delle memorie di testo e caratteri. La Figura 9-4 mostra la disposizione dei 4 banchi da 16K entro i 64K dei banchi 0 e 1.

Ecco come i banchi si dispongono ed operano nel Commodore 128. Un banco da 64K è sempre mappato in memoria (cioè indirizzabile). Da BASIC o da Monitor avete sempre 16 configurazioni disponibili. Per cambiare la configurazione usate il comando BANK con un parametro da 0 a 15 in BASIC e, nel Monitor, precedete l'indirizzo a 4 cifre con una quinta da 0 a F. Al di fuori del BASIC e del Monitor potete selezionare altre configurazioni scrivendo un valore nel registro in $FF00 (o $D500). Andate al Capitolo 14 per i dettagli.

Nella configurazione selezionata, e quindi parte del banco corrente di 64K, esiste un campo riservato di 16K per il banco video. Questo banco di 16K deve racchiudere 1K di memoria di schermo, e 4K di memoria caratteri oppure 8K di bit map. Tutte queste componenti devono essere presenti così da permettere alla grafica di funzionare.

In sintesi, il concetto di banco può essere pensato così: il C128 ha 16K di banco video (VIC) entro una certa configurazione da 64K di RAM.

Le Figure da 9-28 a 9-32 alla fine del capitolo forniscono un sommario della programmazione grafica.

I REGISTRI OMBRA: LOCAZIONI DI MEMORIZZAZIONE INTERMEDIA USATE DALL'EDITOR DI SCHERMO DEL C128

Gli utenti che hanno dell'esperienza nella programmazione del chip VIC del C64 troveranno che molte delle operazioni grafiche sono eseguite nella stessa maniera sul C128. La differenza principale tra i sistemi grafici del C64 e del C128 è l'implementazione hardware dei modi a schermo diviso.

Il modo C128 fornisce due tipi di schermo diviso:

1. Modo carattere e bit map standard
2. Modo carattere standard e bit map multicolore

Poichè gli schermi divisi alternano fra un tipo di schermo ed un altro in un dato tempo, l'editor di schermo deve essere guidato dalle interruzioni.

L'interruzione indica quando bisogna cambiare di modo. A quel punto, si caricano nel chip VIC dei valori prestabiliti in RAM. Questi valori prestabiliti si chiamano registri ombra. Ad ogni occorrenza di un'interruzione, certi registri del VIC sono modificati coi valori dei registri ombra. Questi ultimi rappresentano un cambiamento nella gestione grafica rispetto al Commodore 64.

Le locazioni primarie di memorizzazione intermedia sono:

NOME	LOCAZ. IND.	LOCAZ. DIR.	DESCRIZIONE
GRAPHM	BIT 7 – 216	BIT 4 – 53270	Bit del modo multicolore
GRAPHM	BIT 6 – 216	----------	Bit di schermo diviso
GRAPHM	BIT 5 – 216	BIT 5 – 53265	Bit del modo bit map
VM1*	BIT 7-4 – 2604	BIT 7-4 – 53272	Puntatore alla memoria video
VM1*	BIT 0-3 – 2604	BIT 0-3 – 53272	Puntatore alla memoria carattere
VM2**	BIT 7-4 – 2605	BIT 7-4 – 53272	Puntatore alla memoria video
VM2**	BIT 0-3 – 2605	BIT 0-3 – 53272	Puntatore alla bit map

* VM1 vale solo per i modi di testo standard e multicolore
** VM2 vale solo per la bit map standard e multicolore

Dovete leggere e scrivere attraverso queste locazioni indirette per accedere alle caratteristiche del chip 8564 (VIC). Per esempio, in modo C64, per abilitare la pagina grafica dovete:

10 POKE 53272,120: REM seleziona la bit map @ 8192, la memoria colore @ 7168
20 POKE 53265, PEEK (53265) OR 32: REM abilita la bit map

La linea 10 pone la bit map a 8192 e la memoria colore (la precedente memoria video) a 7168. La linea 20 abilita la pagina grafica.

In modo C128, normalmente fate questa operazione col comando ad alto livello:

GRAPHIC 1

Per fare lo stesso tramite POKE fate come segue:

```
10 POKE 2605,120
20 POKE 216, PEEK (216) OR 32
```

In linguaggio macchina, sul C128:

```
LDA #$78 ;    bit map @ $2000
STA $0A2D;    e memoria video @ 7168
LDA $D8
ORA #$20 ;    bit 5
STA $D8 ;     bit map abilitata
```

Sebbene questi esempi facciano più che selezionare la pagina grafica e stabilire i puntatori del video e della bit map (come aspettare di essere nella zona invisibile del video per cambiare di modo), ciò vi dà un'idea dei passi necessari.

Come avete potuto vedere, il C128 richiede una leggera variazione nella programmazione del chip VIC. Dovete ricordarvene quando programmate in grafica sul C128. Normalmente sono i comandi del BASIC 7.0 a prendersi cura di tutto. Comunque, se state programmando in linguaggio macchina, ricordatevi di riferirvi a queste locazioni piuttosto che a quelle vere. Se infatti scrivete direttamente nei registri del VIC, entro 1/60 di secondo verranno riportati dall'interruzione ai valori dei registri ombra senza cambiare apparentemente di modo grafico.

DISABILITAZIONE DELL'EDITOR DI SCHERMO AD INTERRUZIONI

Potete disabilitare l'editor di schermo ad interruzioni ponendo il valore 255 ($FF) nella locazione 216 ($D8). I veri registri del VIC non vengono influenzati, e diventano disponibili esattamente come in modo 64. Questo rende inutile l'indirizzamento dei registri ombra. Da BASIC fate:

POKE 216,255

In linguaggio macchina invece:

```
LDA #$FF
STA $D8
```

Ricordate, dovete disabilitare le interruzioni o scrivere nei registri ombra, altrimenti difficilmente il vostro programma funzionerà. Le locazioni indirette per il chip a 80 colonne sono trattate nel Capitolo 11, Programmazione del chip a 80 colonne (8563). Pure certe altre funzioni di I/O richiedono l'uso di locazioni indirette. Di queste si parla nel Capitolo 13, Guida all'Input/Output.

IL SISTEMA GRAFICO DEL COMMODORE 128

Questa sezione descrive la posizione delle memorie di schermo, caratteri e bit map nel sistema. Le memorie di schermo e caratteri sono indirizzate differentemente in modo testo ed in modo bit map.

Gli schermi divisi usano una parte di memoria di testo ed una parte di memoria bit map.

Nelle operazioni grafiche il C128 può funzionare indifferentemente sia in BASIC che in linguaggio macchina, e sia in modo C128 che in modo C64.

Questa sezione vi dice la posizione delle locazioni grafiche e delle memorie di schermo, colore e caratteri in ogni modo grafico. La sezione seguente approfondisce il funzionamento di ogni modo grafico, compresi l'interpretazione e l'assegnazione dei dati di testo, carattere e bit map.

MEMORIA DI SCHERMO (RAM)

La locazione della memoria di schermo e il tipo di dato contenuto dipende dal modo grafico corrente del C128.

BASIC C128

Nel BASIC del C128, la memoria di schermo è locata da 1024 ($0400) a 2023 ($07E7). La posizione della memoria di schermo può essere cambiata. Ricordate, alcuni indirizzi usano delle locazioni indirette per cambiare il vero indirizzo. Il registro ombra del puntatore allo schermo è locato a 2604 ($0A2C). La vera locazione sarebbe 53272, ma questa viene continuamente caricata da 2604 durante le interruzioni. Una POKE diretta a 53272 viene annullata in 1/60 di secondo. Ecco come cambiare la locazione della memoria di schermo dal BASIC del C128:

POKE 2604, (PEEK (2604) AND15) OR X

Dove X è un valore preso dalla Figura 9-6. Se muovete la memoria di schermo, assicuratevi che non si sovrapponga alla memoria caratteri. Inoltre, ricordatevi di aggiungere uno spostamento di $4000 (16384) alle memorie di schermo e caratteri per ogni banco video. Occorrono comunque altri comandi perchè il programma funzioni. I dettagli seguiranno nella trattazione dei singoli modi grafici come anche negli esempi di programma.

		LOCAZIONI*	
X	**BIT**	**DECIMALE**	**ESADECIMALE**
0	0000 - - - -	0	$0000
16	0001 - - - -	1024	$0400(Default)
32	0010 - - - -	2048	$0800
48	0011 - - - -	3072	$0C00
64	0100 - - - -	4096	$1000
80	0101 - - - -	5120	$1400
96	0110 - - - -	6144	$1800
112	0111 - - - -	7168	$1C00
128	1000 - - - -	8192	$2000
144	1001 - - - -	9216	$2400
160	1010 - - - -	10240	$2800
176	1011 - - - -	11264	$2C00
192	1100 - - - -	12288	$3000
208	1101 - - - -	13312	$3400
224	1110 - - - -	14336	$3800
240	1111 - - - -	15360	$3C00

* Ricordatevi di aggiungere uno spostamento di $4000 per ogni banco video superiore allo 0.

Figura 9-6. Locazioni della memoria di schermo

Questo registro controlla pure la posizione della memoria caratteri. I 4 bit alti controllano lo schermo, i 4 bassi la memoria carattere. L'AND 15 nell'istruzione POKE 2604 garantisce che il nybble (semibyte) più basso non sia alterato, altrimenti spostereste anche la memoria caratteri.

Nel modo bit map (standard o multicolore) del Commodore 128, la memoria di schermo (memoria colore in questo caso) è locata da 7168 ($1C00) a 8167 (1FFF). L'interpretazione della memoria di schermo è differente in modo bit map: essa fornisce infatti non i caratteri, ma il colore per ogni posizione. Ciò è spiegato in dettaglio nella sezione del Modo Bit Map Standard, altrove in questo capitolo. Per cambiare la locazione della memoria di schermo della bit map usate questo comando:

POKE 2605, (PEEK (2605) AND 15) OR X

dove X è un valore come da Figura 9-6. Pure la locazione 2605 è un registro ombra per 53272, ma solo per il modo bit map. Quando spostate la memoria video, assicuratevi che non si sovrapponga alla memoria bit map. Ricordatevi ancora di aggiungere uno spostamento di $4000 per ogni banco video superiore allo 0.

BASIC C64

In modo C64, lo schermo di testo va da 1024 ($0400) a 2023 ($07E7), così come in modo bit map, pur cambiandone l'interpretazione. Il BASIC del Commodore 64 vi permette di muovere la matrice video in una qualsiasi delle 16 posizioni specificate in Figura 9-6. Pure qui i 4 bit alti di 53272 ne controllano la posizione. Per cambiare la posizione della memoria di schermo, date questo comando:

POKE 53272, (PEEK(53272) AND 15) OR X

dove X va preso sempre da Figura 9-6.

NOTA: il paragrafo seguente si riferisce ad entrambi i modi C64 e C128.

I bit 0 e 1 della locazione 56576 ($DD00) controllano quale dei 4 banchi video è usato. Il banco di default è lo 0. Se passate in un qualunque altro banco maggiore dello 0, aggiungete uno spostamento di $4000 (16384) per ogni banco. Ciò produrrà il vero indirizzo della matrice video. Per esempio, se cambiate dal banco 0 al banco 1, aggiungete $4000. Se andate nel banco 2, $8000; nel 3, $C000. Ricordate che ciò vale sia per i modi 64 e 128.

LINGUAGGIO MACCHINA

In linguaggio macchina, usate i comandi listati sotto A in Figura 9-7 per spostare la matrice video in modo 128, quelli sotto B per spostare la bit map, sempre in modo 128, e quelli sotto C per il modo 64 (sia memoria di testo o colore).

(A)	(B)	(C)
SCHERMO DI TESTO C128	**SCHERMO BIT MAP C128**	**SCHERMO DI TESTO O BIT MAP C64**
LDA $0A2C	**LDA $0A2D**	**LDA $D018**
AND #$0F	**AND #$0F**	**AND #$0F**
ORA #$X	**ORA #$X**	**ORA #$X**
STA $0A2C	**STA $0A2D**	**STA $D018**

Figura 9-7. Spostamento della memoria di schermo da linguaggio macchina

Nella Figura 9-7, X è l'equivalente esadecimale del valore di X della colonna più a sinistra di Figura 9-6. La seconda e la terza istruzione di ogni esempio in Figura 9-7 conservano lo stato dei 4 bit bassi della locazione 53272 o dei corrispondenti registri ombra in 2604 ($0A2C) e 2605 ($0A2D), per non spostare le memorie carattere e bit map.

RAM COLORE

BASIC C128

La RAM colore nel Commodore 128 va sempre da 55296 ($D800) a 56295 ($DBE7), e non è possibile spostarla. In modo carattere standard ogni locazione della memoria di schermo e della memoria colore corrisponde univocamente. Così la locazione 1024 prende il colore da 55296, la 1025 da 55297 e così via. Anche il modo carattere multicolore utilizza la RAM colore, ma in un modo differente. Ulteriori spiegazioni ed esempi saranno forniti nella sezione del Modo Carattere Standard di questo capitolo.

SUDDIVISIONE A BANCHI DELLA RAM COLORE

In modo 128, le linee di segnale LORAM e HIRAM permettono al sistema grafico di fare uso di un ulteriore banco di RAM colore, non disponibile però in modo 64. Ciò permette di alternare i colori dal modo carattere al modo bit map in modo veloce e pulito. Il segnale LORAM controlla l'accesso dell'8502 ad uno dei 2 banchi di RAM colore, mentre HIRAM regola l'accesso del chip VIC, indipendentemente dall'8502. I segnali LORAM e HIRAM sono controllati rispettivamente dai bit 0 e 1 della locazione 1. LORAM rende accessibile per l'8502 il banco colore 0 o 1 a seconda del valore del bit: 0=banco 0, 1=banco 1. HIRAM invece esegue lo stesso compito ma per il chip VIC: 0=banco 0, 1=banco 1.

Queste linee di controllo aggiungono flessibilità al già potente sistema grafico del C128. Questo vi permette di cambiare al volo i colori del modo bit map multicolore o dello schermo di testo, senza alcun ritardo. Lo scambio fra banchi di RAM è istantaneo.

In modo bit map standard, le informazioni di colore sono ottenute dalla memoria di schermo locata da 7168 ($1C00) a 8167 ($1FFF), non dalla memoria colore. Il modo bit map interpreta la memoria colore diversamente dal modo carattere. La RAM colore viene usata nei modi carattere standard, bit map multicolore, carattere multicolore e schermo diviso.

BASIC C64

In modo carattere standard, la RAM colore è ancora locata da 55296 a 56295. In modo bit map, il chip VIC riceve le informazioni sul colore esattamente come in modo 128, soltanto che la memoria video è locata da 1024 a 2023.

LINGUAGGIO MACCHINA

Sia in linguaggio macchina che in BASIC, il modo carattere standard riceve le informazioni sul colore dalla RAM colore, usata sia in modo standard che multicolore. In modo bit map standard invece il colore proviene dalle locazioni della memoria di schermo, ovunque questa sia locata. Infine, in modo bit map multicolore, i colori vengono presi dalla memoria di schermo, la RAM colore e il registro 0 del colore di fondo (53281). Ciò è spiegato in profondità nelle sezioni dei modi carattere e bit map multicolore.

MEMORIA CARATTERI (ROM)

BASIC C128 – MODI CARATTERE

In modo carattere standard, l'informazione sui caratteri è posta in memoria da 53248 ($D000) a 57343 ($DFFF). Il segnale CHAREN (CHARacter ENable, abilitazione caratteri), controllato dal bit 2 della locazione 1, abilita o meno l'immagine della ROM caratteri nel banco video corrente. Se il bit 2 vale 0, allora appare l'immagine della ROM da 4096 a 8191 nel banco video corrente; viceversa, un valore 1 nasconde l'immagine della ROM facendo apparire la RAM. Chiariamo che l'immagine della ROM è un artificio usato perchè il chip VIC, che non indirizza oltre i 16K, possa prelevare le informazioni sui caratteri in ogni banco video, anche se la ROM è realmente locata a 53248. Per l'8502, in effetti, una lettura o una scrittura da 4096 a 8191, in ogni banco video, interessa sempre la RAM. Se vogliamo quindi leggere la ROM caratteri, dobbiamo usare la configurazione 14, col comando BANK od il prefisso E nel monitor, poi leggere da 53248. La Figura 9-8 mostra la disposizione dei set di caratteri nella ROM:

	INDIRIZZI		IMMAGINE	CONTENUTI
BLOCCO	**DECIM.**	**ESA**		
	53248	**D000**	**$1000**	**Maiuscole**
0	**53760**	**D200**	**$1200**	**Grafiche**
	54272	**D400**	**$1400**	**Maiuscole negative**
	54784	**D600**	**$1600**	**Grafiche negative**
	55296	**D800**	**$1800**	**Minuscole**
1	**55808**	**DA00**	**$1A00**	**Maiuscole e grafiche**
	56320	**DC00**	**$1C00**	**Minuscole negative**
	56832	**DE00**	**$1E00**	**Maiuscole e grafiche negative**

Figura 9-8

Pure la memoria carattere è rilocabile come la memoria di schermo. Sul C128, per spostarla, alterate i 4 bit bassi della locazione 2604 ($0A2C). Quest'ultima è il registro ombra per 53272, relativo alla memoria di schermo (4 bit alti) e alla memoria carattere (4 bit bassi). Usate questo comando per muoverne la posizione:

POKE 2604, (PEEK (2604) AND 240) OR Z

dove Z è un valore di Figura 9-9.

LOCAZIONI DELLA MEMORIA CARATTERE

Z	BIT	DECIM.	ESA
0	----000-	0	$0000
2	----001-	2048	$0800
4	----010-	4096	$1000 Immagine della ROM in ogni banco
6	----011-	6144	$1800 Immagine della ROM in ogni banco
8	----100-	8192	$2000
10	----101-	10240	$2800
12	----110-	12288	$3000
14	----111-	14336	$3800

Figura 9-9. Locazioni della memoria carattere

Come gli altri componenti del sistema grafico, i dati dei caratteri si comportano differentemente nei modi testo e bit map.

Ricordatevi di non alterare i 4 bit alti, che controllano la memoria di schermo. L'AND240 nell'istruzione POKE serve appunto a conservarne lo stato.

In modo 128 il set di caratteri appare in ogni banco video a seconda della linea CHAREN.

> **NOTA:** Ricordate di aggiungere uno spostamento di $4000 anche alla memoria carattere, per ogni banco maggiore di 0; per esempio, nel banco 3: 3*$4000=$C000.

BASIC C128 — MODI BIT MAP

In modo bit map, la memoria carattere, chiamata ora bit map, parte da 8192 e continua fino a 16191, per 8000 byte. La configurazione di bit di ognuno di questi 8000 byte determina l'accensione o meno di determinati pixel, secondo la relazione 1=acceso, 0=spento. Poichè la risoluzione è di 320x200, ci sono 64000 punti rappresentabili sullo schermo. Assegnando un bit ad ogni punto e dividendo per 8, arriviamo agli 8000 byte necessari.

Accanto ai 4 bit alti di 2605, solo il bit 3 (valore 8) è significativo per la bit map.

Quando date il comando **GRAPHIC** 1, il bit 3 di 2605 viene posto a 1. Ciò fa sì che la bit map parta da 8192. Ogni qualvolta date il comando GRAPHIC, il bit 3 di 2605 è sempre messo a 1. Se, al di fuori del BASIC, volete far partire la bit map da $0000, azzerate il bit 3.

Se il C128 sta girando sotto il controllo del BASIC, la bit map è sempre situata oltre i primi 8192 byte nel banco video corrente. Nei banchi 1, 2 e 3 parte rispettivamente da $6000, $A000 e $E000, a causa dello spostamento di $4000 per ogni banco. In linguaggio macchina, invece, il bit 3 può valere 0 o 1, a vostra scelta. Ciò implica che potete far partire la bit map da $0000 o da $2000 in ognuno dei 4 banchi video, per un totale quindi di 8 possibili locazioni per ogni banco da 64K.

In BASIC, al contrario, poichè il bit 3 viene sempre messo a 1, avete solo 4 possibili locazioni d'inizio, sempre per ogni banco da 64K.

Non dimenticate di aggiungere obbligatoriamente $4000 per ogni banco video. Le 8 possibili locazioni di partenza sono mostrate in Figura 9-10.

BANCO VIDEO	VALORE DEL BIT 3	
	0	1
0	$0000	$2000
1	$4000	$6000
2	$8000	$A000
3	$C000	$E000

Figura 9-10. Locazioni della bit map possibili in linguaggio macchina

Sebbene in ogni banco video possano trovar posto fino a 2 pagine grafiche, dovete lasciar spazio anche per la memoria di schermo. Ogni banco video ha infatti 16K, di cui dovete riservarne 9 per ogni bit map (8K + 1K). Appare chiaro dunque che in ogni banco da 64K potete avere solo 4 bit map, che arrivano a 8 usando tutti i 128K. Per locare una bit map nella RAM del banco 1, ricordate di cambiare anche il registro di configurazione dell'MMU.

BASIC C64 — MODI CARATTERE

In modo carattere standard, i 4 bit bassi di 53272 controllano la locazione della memoria carattere. Cosí come in modo 128, la ROM caratteri è in realtà locata da 53248, mentre una sua immagine appare SOLO nei banchi video 0 e 2, rispettivamente da 4096 a 8191 e da 36864 a 40959. L'immagine scompare completamente nei banchi 1 e 3. Questo concetto si applica solamente al chip VIC, mentre l'8502 legge e scrive normalmente dalla RAM.

In modo 64, l'immagine della ROM ricopre la RAM sottostante. Un'operazione di scrittura scriverà nella RAM sottostante, mentre una di lettura ritornerà un valore della ROM a seconda della configurazione corrente. Poiché il chip VIC accede solo a 16K contemporaneamente, le immagini del set di caratteri devono apparire nei 16K che il chip VIC sta indirizzando attualmente nei banchi 0 e 2.

In modo 128, invece, il bit 4 (CHAREN) della locazione 1 abilita o disabilita l'immagine in TUTTI e 4 i banchi video.

Potete cambiare la locazione della memoria carattere col comando:

POKE 53272, (PEEK(53272) AND 240) OR Z

dove Z è un valore decimale di Figura 9-9.

La disposizione dei set di caratteri è identica al modo 128 (osservate la Figura 9-8).

BASIC C64 — MODO BIT MAP

In modo bit map, il bit 3 di 53272 determina l'inizio della bit map a $0000 o $2000, a seconda se sia 0 o 1. Usate il comando seguente:

POKE 53272, (PEEK(53272) AND 240) OR Z

dove naturalmente Z vale 0 per iniziare a $0000, e 8 per iniziare a $2000 (8192), in ogni banco video.

Osservate la Figura 9-10 per le disposizioni della bit map in ognuno dei 4 banchi video da 16K entro i 2 banchi da 64K. Se cambiate di banco video, non dimenticate di aggiungere lo spostamento di $4000. Andate alla parte della Memoria Caratteri nella sezione relativa al Modo Bit Map per ulteriori spiegazioni sulla disposizione in memoria della bit map.

LINGUAGGIO MACCHINA

Ci sono tre modi di selezionare la posizione in memoria della mappa caratteri, come mostrato in Figura 9-11. L'esempio A usa il registro ombra 2604. Il B fa partire la bit map da $2000 usando il registro 2605. Il C infine specifica l'inizio della memoria carattere o della bit map in modo 64.

In Figura 9-11, Z è un valore preso da Figura 9-9.

A	B	C
LDA $0A2C	LDA $0A2D	LDA $D018
AND #$F0	AND #$F0	AND #$F0
ORA #$Z	ORA #$08	ORA #$Z
STA $0A2C	STA $0A2D	STA $D018

Figura 9-11. Selezione delle locazioni della memoria carattere

MODO CARATTERE STANDARD

COME SELEZIONARE IL MODO CARATTERE STANDARD

Il C128 all'accensione si regola in modo carattere standard. Questo modo mostra i caratteri sullo schermo di default, un colore per ogni carattere su uno sfondo unico. Questo è il modo in cui voi scrivete i programmi, e che potete sempre ristabilire premendo RUN/STOP+RESTORE.

La locazione 53265 (e il suo registro ombra $D8) determina se il C128 funziona in modo testo o bit map. Il bit 5 di $D8 vale 0 in modo testo (come all'accensione) e 1 in modo bit map.

La locazione 53270 (e il suo registro ombra $D8) determina il modo standard o multicolore. Il bit 4 di 53270, ed il suo bit ombra, il bit 7 di $D8, valgono 0 in modo standard e 1 in multicolore. Andate alla sezione del Modo Carattere Multicolore per maggiori dettagli.

LOCAZIONI DI SCHERMO

In modo carattere standard, la memoria di schermo va normalmente da 1024 a 2023, ed è rilocabile. Per la rilocazione, andate alla sezione della Memoria di Schermo nelle pagine precedenti.

Poichè lo schermo è composto di 40 colonne per 25 righe, sono necessarie 1000 locazioni per conservare le informazioni sui caratteri. Le ultime 24 locazioni non contengono dati visualizzati, ma vengono usate per altri scopi. Ogni carattere che vedete sullo schermo ha la sua propria locazione di memoria. La locazione in alto a sinistra, chiamata HOME (casa), ha l'indirizzo 1024. La seconda, a destra, il 1025, e così via. Sebbene lo schermo che vedete sia composto di righe e colonne, la memoria di schermo è lineare, cioè parte da 1024 e finisce a 2023, senza suddivisioni di sorta.

La Figura 9-12 mostra la mappa della memoria di schermo, così da rilevare la corrispondenza fra memoria e caratteri visualizzati.

MAPPA DI MEMORIA DELLO SCHERMO

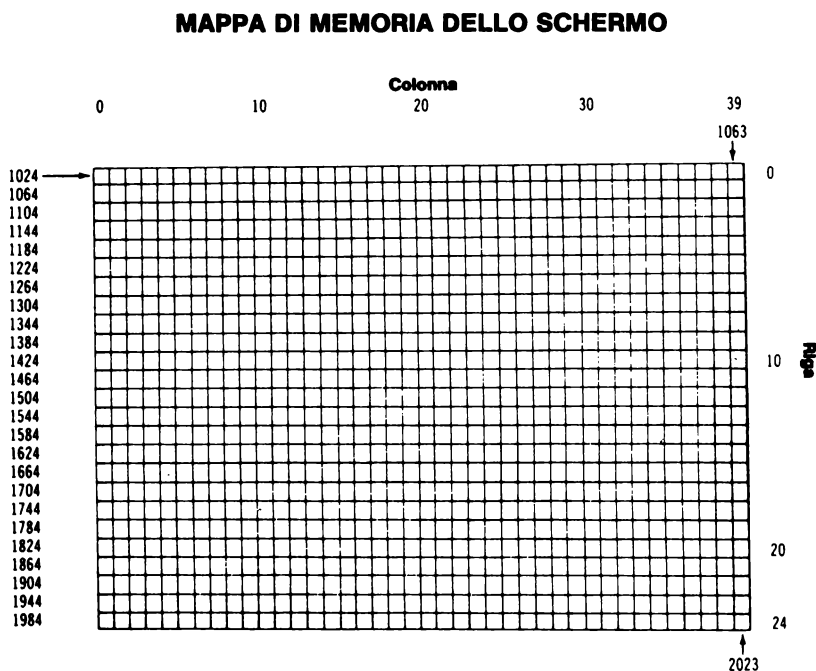



Figura 9-12. Mappa di memoria dello schermo

INTERPRETAZIONE DELLA MEMORIA DI SCHERMO

Questo tipo di memoria conserva solo interi caratteri. Ogni locazione contiene il codice relativo, che non è quello ASCII, ma un codice di schermo come quelli raccolti nell'Appendice D, Volume 8. La differenza è dovuta alla disposizione dei caratteri nella ROM. Notate nell'Appendice D che il codice di schermo per @ è 0. @ ha il codice 0 perchè è il primo carattere nella ROM. La lettera A è il secondo carattere, quindi ha codice 1, la B è il terzo, codice 2, e così via. Il codice di schermo è in realtà un indice che punta alla ROM caratteri partendo da 0.

Se volete memorizzare direttamente, con POKE, un carattere sullo schermo, usate il codice di schermo piuttosto che il codice CHR$. Lo stesso vale per il Monitor. Per esempio:

```
POKE 1024,1
```

pone la lettera A nella posizione HOME dello schermo a 40 colonne. Dal monitor ottenete lo stesso effetto ponendo 1 in $0400 (1024).

DATI DEI COLORI

Nel modo carattere standard, le informazioni sul colore provengono dalla RAM da 55296 ($D800) a 56295 ($DBE7). Questa memoria determina il colore di ognuno dei 1000 caratteri. Il colore di fondo proviene invece dal registro di fondo 0, in 53281.

Le locazioni della RAM colore e di schermo corrispondono su una base uno per uno. La locazione di schermo in 1024 fa riferimento alla locazione di RAM colore in 55296; la locazione 1025 corrisponde alla locazione colore 55297, ecc. La Figura 9-13 è la mappa della RAM colore e mostra la corrispondenza fra colore del carattere e memoria.

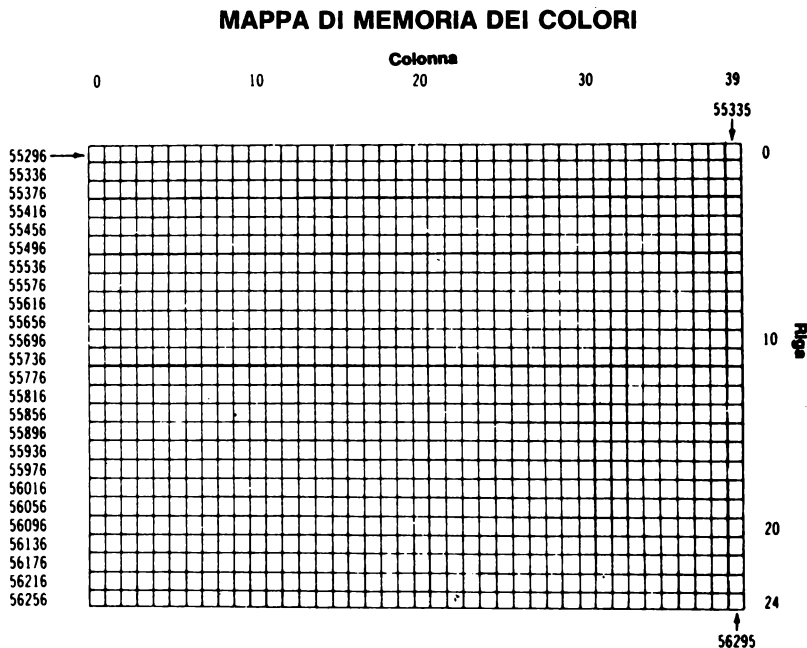



Figura 9-13. Mappa di memoria dei colori

INTERPRETAZIONE DELLA MEMORIA COLORE

Le locazioni della RAM colore contengono un codice da 0 a 15 (i 4 bit alti vengono ignorati) corrispondente ai seguenti colori:

0 Nero	**8 Arancio**
1 Bianco	**9 Marrone**
2 Rosso	**10 Rosso chiaro**
3 Azzurrino	**11 Grigio scuro**
4 Porpora	**12 Grigio medio**
5 Verde	**13 Verde chiaro**
6 Blu	**14 Azzurro**
7 Giallo	**15 Grigio chiaro**

Figura 9-14. Codici dei colori a 40 colonne

Notate che questi codici corrispondono a quelli usati da tastiera e dal BASIC — 1. Se volete quindi porre un colore direttamente in memoria, usate i codici della tabella 9-14, non quelli usati da tastiera e BASIC. Per esempio:

POKE 55296, 1

colora il carattere in posizione HOME di bianco. Da monitor, ponete 1 in $D800 per avere lo stesso risultato.

Ricordate che questi codici controllano solo il colore del carattere. Il colore del fondo proviene dal registro di fondo 0 (53281). I pixel del carattere corrispondono ai bit dei byte che lo compongono nella mappa caratteri: un bit a 1 accende il pixel corrispondente, un bit a 0 lo spegne; i pixel accesi sono del colore nella RAM colore, i pixel spenti invece del colore di fondo in 53281. Imparerete di più sulla composizione dei caratteri nei prossimi paragrafi.

MEMORIA DEI CARATTERI

In modo carattere standard, il chip VIC riceve le informazioni sui caratteri dalla ROM caratteri. Quest'ultima va da 53248 a 57343. Poichè il chip VIC può indirizzare solo 16K, occorre un artificio per rendere visibile la ROM dei caratteri in QUEI determinati 16K. In modo 128, la ROM è disponibile in ognuno dei 4 banchi video, a seconda della linea CHAREN. Andate alla sezione della Memoria Caratteri su ROM, più indietro.

In modo 64, la ROM caratteri è disponibile solo nei banchi 0 e 2 rispettivamente da 4096 a 8191 ($1000—1FFF) e da 36864 a 40959 ($9000—9FFF), come immagine della vera ROM, che parte invece da 53248. Nei banchi 1 e 3 invece il VIC vede solo la RAM.

Notate che la ROM caratteri occupa lo stesso spazio dell'I/O (53248—57343), ma non allo stesso tempo. Quando il chip VIC accede alla ROM caratteri, questa viene riprodotta come immagine nel banco video corrente (solo in modo 64), in modo da ricadere entro i 16K (in pratica nello spazio da 53248 a 57343 l'I/O é disponibile solo quando il chip VIC non accede alla ROM caratteri). Essendo un'immagine non danneggia la RAM sottostante, che quindi è sempre a disposizione del processore 8502. Le locazioni della memoria carattere possono essere spostate. Andate alla sezione della Memoria Caratteri su ROM più indietro.

L'INTERPRETAZIONE DELLA MEMORIA CARATTERI IN MODO CARATTERE STANDARD

Di solito, un completo set di caratteri ne contiene 256. Il C128 contiene 2 set di caratteri, per un totale di 512 caratteri. Con lo schermo a 40 colonne (VIC), è disponibile solo un set alla volta, e precisamente il maiuscolo—grafico (condizione di accensione). Per accedere al set maiuscolo—minuscolo premete contemporaneamente i tasti **Commodore** e SHIFT. Nella ROM caratteri, ogni carattere richiede 8 byte per la sua composizione. Poichè 256*8 fa 2048 byte, cioè 2K, e dacchè il C128 ha 2 set di caratteri, in totale ci sono 4K di ROM. La Figura 9-15 mostra la composizione della ROM caratteri.

	INDIRIZZI			
BLOCCO	**DECIM.**	**ESA**	**IMMAGINE**	**CONTENUTI**
	53248	D000	$1000	Maiuscole
0	53760	D200	$1200	Grafiche
	54272	D400	$1400	Maiuscole negative
	54784	D600	$1600	Grafiche negative
	55296	D800	$1800	Minuscole
1	55808	DA00	$1A00	Maiuscole e grafiche
	56320	DC00	$1C00	Minuscole negative
	56832	DE00	$1E00	Maiuscole e grafiche negative

Figura 9-15. Set di caratteri in ROM

Notate che in realtà il C128 possiede 8K di ROM — 4K per il modo 64 e 4K per il modo 128. Il sistema seleziona automaticamente l'appropriata ROM per ogni modo operativo.

Le disposizioni di bit memorizzate nella ROM caratteri sono in diretta relazione con i pixel sullo schermo. In memoria, ogni carattere richiede 8 byte. Sullo schermo, un carattere è formato da una matrice di 8x8 pixel. Pensate ad un carattere come 8 righe di 8 pixel ciascuna. Ogni riga di pixel richiede un byte di memoria, così ogni pixel richiede un bit.

Poichè un carattere è una matrice di 8x8 pixel, richiederà un totale di 64 pixel o 8 byte. Entro ogni byte, se un bit vale 1, allora il pixel corrispondente sarà acceso. Viceversa, se un bit vale 0, il pixel sarà spento. La combinazione di pixel accesi e

spenti forma il carattere sul video. La Figura 9-16 dimostra la corrispondenza fra un carattere sullo schermo e la corrispondente rappresentazione nella ROM caratteri.

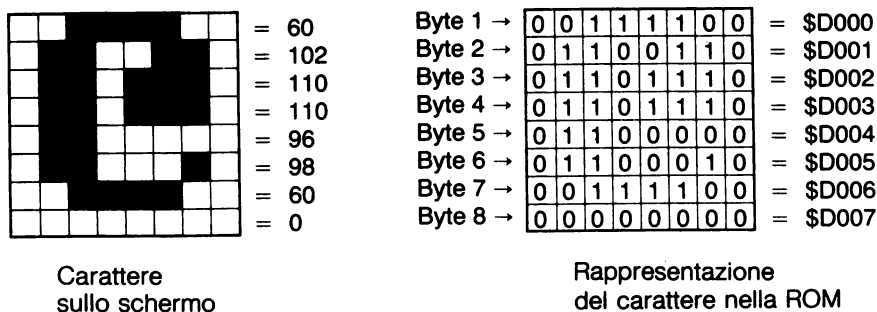



Figura 9-16. Relazione fra carattere sullo schermo e nella ROM

In Figura 9-16, i primi 8 byte della ROM caratteri ($D000 – D007) hanno i valori 60, 102, 110, 110, 96, 98, 60 e 0. Questi valori decimali sono ricavati dai valori binari degli 8 byte relativi alle righe di pixel che formano il carattere. Per ogni bit di valore 1 si somma la potenza di 2 associata. Per esempio, il primo byte è uguale a 60, calcolato sommando i valori delle seguenti posizioni di bit:

$2^5 + 2^4 + 2^3 + 2^2 = 32 + 16 + 8 + 4 = 60$

I bit che sono a 1 corrispondono ai pixel accesi, cioè a quelli del colore di primo piano. Viceversa i bit a 0 sono i pixel spenti, visualizzati nel colore di fondo preso dal registro 0 in 53281.

Il secondo byte (riga di pixel) di @ è uguale a 102 (decimale) ed è ottenuto come segue:

$2^6 + 2^5 + 2^2 + 2^1 = 102$

L'ultimo byte di @ vale 0, perchè nessun pixel è acceso. Quindi, ogni pixel sullo schermo viene mostrato nel colore di fondo. I valori dei bit sulla destra di Figura 9-16 sono in corrispondenza diretta coll'immagine del carattere così come appare sulla sinistra della stessa figura.

ACCESSO ALLA ROM DEI CARATTERI

BASIC C128

Per accedere alla ROM dei caratteri dal BASIC del C128, usate il seguente programma:

```
10  BANK 14
20  FOR I=53248 TO 53248+7:PRINTPEEK(I);:NEXT
30  BANK 15
```

Entra nel banco 14, l'unico banco BASIC in cui si può accedere alla ROM caratteri. Poi stampa il contenuto dei primi 8 byte della ROM. Una volta finito, torna al banco 15.

LINGUAGGIO MACCHINA

Per accedere in linguaggio macchina alla ROM caratteri, usate quest'altro programma:

```
. 01800 A9 01      LDA #$01
. 01802 8D 00 FF   STA $FF00
. 01805 A2 00      LDX #$00
. 01807 BD 00 D0   LDA $D000,X
. 0180A 9D 40 18   STA $1840,X
. 0180D E8         INX
. 0180E E0 07      CPX #$07
. 01810 D0 F5      BNE $1807
. 01812 A9 00      LDA #$00
. 01814 8D 00 FF   STA $FF00
. 01817 60         RTS
```

```
10  SYS 6144
20 FOR I=6208 TO 6208+7:PRINTPEEK(I);:NEXT
```

Queste due routine in BASIC e linguaggio macchina eseguono lo stesso compito del primo programma BASIC. Le prime due istruzioni in linguaggio macchina abilitano la ROM caratteri disabilitando l'I/O. Le 6 istruzioni successive trasferiscono i primi 8 byte della ROM caratteri nelle locazioni da 6208 ($1840) a 6215 ($1847). Le ultime tre istruzioni riabilitano l'I/O e ritornano al BASIC.

La routine BASIC chiama la subroutine in linguaggio macchina, poi stampa i valori memorizzati temporaneamente da 6208 a 6215. Andate al Capitolo 7, Immissione dei programmi in linguaggio macchina, per i dettagli sulla memorizzazione delle istruzioni di linguaggio macchina sul C128.

BASIC C64

Per accedere alla ROM caratteri in modo 64, usate quest'ultimo programma:

```
40  POKE 56334,PEEK(56334) AND 254
50  POKE 1,PEEK (1) AND 251
80  FOR I=0 TO 7:POKE 6144+I, PEEK(53248+I):NEXT
90  POKE 1,PEEK(1) OR 4
105 POKE 56334,PEEK(56334) OR 1
130 FOR I=6144 TO 6144+7:PRINTPEEK(I);:NEXT
```

La linea 40 ferma il timer A, il generatore delle interruzioni. La linea 50 disabilita l'I/O ed abilita la ROM caratteri. La linea 80 trasferisce i primi 8 byte della ROM (53248–53255) da 6144 a 6151. La linea 90 disabilita la ROM e riabilita l'I/O. La linea 105 riavvia il timer A. La linea 130 stampa i primi 8 byte della ROM prendendoli da 6144 a 6151.

Potreste voler trasferire alcune parti della ROM caratteri in RAM durante la creazione del vostro proprio set di caratteri, volendo mantenere una parte dei caratteri della ROM. Ciò sarà spiegato in dettaglio più avanti nel capitolo. Per ora abbiamo mostrato come e perchè accedere alla ROM caratteri e la codifica dei caratteri da pixel a bit.

La prossima sezione spiegherà come ridefinire i caratteri in modo C128.

CARATTERI RIDEFINIBILI

Il Commodore 128 ha una caratteristica che vi permette di ridisegnare il set di caratteri secondo le vostre necessità. In molti casi, vorrete ridefinire solo una parte del set, traendo i caratteri restanti dalla ROM.

Usando i caratteri programmabili, voi dite al C128 di prendere le informazioni sui caratteri dalla RAM invece che dalla ROM. Potete anche prelevare solo alcuni caratteri dalla ROM ed usarli in RAM assieme ai vostri. Ricordate che non potete scrivere su ROM, quindi per modificare il set dovete prima ricopiarlo su RAM, poi fare le vostre modifiche.

Il primo passo nella programmazione dei vostri caratteri è definirne l'immagine. Nella sezione del Modo Carattere Standard, avete visto il modo in cui un carattere è memorizzato in ROM. Ogni carattere richiede 8 byte di memoria. Ogni byte corrisponde ad una riga di pixel entro la matrice 8x8 del carattere; quindi, 8 righe di pixel formano un carattere.

Questa sezione mostrerà la creazione di un set di caratteri maiuscolo corsivo per le lettere dalla A alla H. La Figura 9-17 mostra il disegno della A maiuscola corsiva. La griglia nella figura visualizza il modo in cui appare un carattere nella

matrice di 8x8 pixel. Ogni riga della griglia determina quali bit sono a 1 nella maschera del carattere e quindi quali pixel sono accesi sullo schermo. Le stringhe binarie di 8 bit sulla destra sono la rappresentazione in RAM. I numeri a destra delle stringhe sono l'equivalente esadecimale delle stringhe. Questi valori decimali sono i dati che inserite in RAM tramite POKE.

Figura 9-17. Disegno della lettera A corsiva

Il programma seguente crea e mostra le lettere maiuscole corsive dalla A alla H. Immettetelo nel computer ed eseguitelo. Vedrete le lettere da A a H cambiare dal set a blocchi al set corsivo. Quando premete i tasti corrispondenti, vedrete le lettere nella nuova forma corsiva.

```
10 PRINTCHR$(142):REM SELEZIONA IL SET MAIUSCOLO
20 POKE4627,48:REM FINE DEL BASIC A 12288
30 BANK14:REM ROM CARATTERI DA 53248
40 FORI=0TO511:POKEI+12288,PEEK(I+53248):NEXT:REM TRASFERIMENTO DA ROM A RAM
50 BANK15:REM I/O DA 53248
60 POKE2604,(PEEK(2604)AND240)+12:REM MAPPA CARATTERI DA 12288
70 FORJ=12288TO12288+71:REM PONE IN RAM I DATI DEI CARATTERI
80 READA
90 POKEJ,A
100 NEXT
110 SCNCLR
120 DATA0,0,0,0,0,0,0,0:REM @
130 DATA14,17,32,66,130,132,138,123:REM A
140 DATA124,66,66,124,66,81,225,126:REM B
150 DATA62,67,130,128,128,128,131,126:REM C
160 DATA125,98,125,65,65,193,161,254:REM D
170 DATA225,66,64,56,120,65,66,124:REM E
180 DATA127,129,2,4,14,228,68,56:REM F
190 DATA193,163,253,33,249,65,99,190:REM G
200 DATA227,165,36,36,126,36,37,231:REM H
```

La linea 10 seleziona il set maiuscolo da ridefinire. La 20 protegge il set dalla sovrascrittura da parte del programma BASIC riservando una zona di memoria da 12288 e purtroppo riducendo di molto la zona disponibile per i programmi. Il set verrà posto da 12288 fino a 16383 (ma non deve necessariamente stare lí), entro cioè il banco video 0 (a meno che non abbiate selezionato un altro banco video). Ogni banco consiste di 16K, poiché il VIC indirizza solo 16K. Potreste lasciare un'area maggiore per il programma BASIC, ma ciò comporterebbe l'uso di un banco video maggiore di 0, con uno spostamento di $4000 per banco. Questo programma di esempio opera nel banco 0. Un trucco più semplice consiste nell'usare l'istruzione GRAPHIC, che alloca lo spazio per la pagina grafica trasferendo automaticamente più in alto il programma BASIC. Se diamo infatti GRAPHIC 1 e subito dopo GRAPHIC 0 (o GRAPHIC 5 se usiamo il video a 80 colonne) il programma sarà trasferito da 16384 in poi, e la zona da 7168 a 16383 resterà intoccabile e libera per i nostri set: infatti in 9K possiamo memorizzarne fino a 4, con 1K di avanzo.

La linea 30 seleziona il banco 14. Questa configurazione rende visibile la ROM caratteri tramite PEEK o da Monitor, escludendo temporaneamente l'I/O. Sia l'I/O che la ROM caratteri condividono le stesse locazioni. A seconda dello stato del bit 0 del registro di configurazione ($FF00) il 128 indirizza i registri di I/O o la ROM caratteri. All'accensione il bit 0 é a zero, quindi nelle locazioni $D000 – DFFF sono visibili i registri di I/O. Il comando BANK nella linea 30 pone a 1 il bit 0 di $FF00; a questo punto in $D000 – DFFF appare la ROM caratteri, a cui si accede per trasferire in RAM i caratteri.

La linea 40 esegue il vero trasferimento da ROM a RAM. La ROM caratteri ora comincia da 53248. Vengono ricopiati i primi 512 byte (il set maiuscolo) dalla ROM alla RAM nelle locazioni da 12288 a 12800. A questo punto il set di caratteri é pronto per essere ridefinito.

La linea 50 ripristina i registri di I/O disabilitando la ROM caratteri.

La linea 60 specifica l'inizio della memoria caratteri. Non è comunque obbligatorio farla partire da 12288: nella sezione sulla Memoria Caratteri è spiegato il posizionamento della memoria. La locazione 2604 è il registro intermedio usato dall'editor del C128 per localizzare le memorie di testo e caratteri. Dovete usare per forza questa locazione invece di 53272, altrimenti entro 1/60 di secondo l'interruzione rieguaglierà 53272 a 2604. Nella sezione sui Registri Ombra è comunque spiegato come disabilitare l'editor di schermo ed accedere direttamente ai registri video.
Il valore (AND240) OR 12 viene posto in 2604 per dire al 128 di localizzare la memoria caratteri a 12288. Potete così avere fino a 4K (2 set) di caratteri ridefiniti. Se però vi accontentate di 1K soltanto potete allocare il vostro set da 6144 a 7167, senza spostare il programma BASIC. Nel caso allochiate lí il set, dovreste dire al 128 di vedere in quella zona la RAM invece dell'immagine della ROM; fate dunque POKE 217,4 ed abiliterete la RAM; poi POKE 2604,22 e sposterete la mappa caratteri a 6144.

Le linee da 70 a 100 eseguono il ciclo da 12288 a 12359 (linee 70-100) di lettura (linea 80) delle DATA (linee 120-200) ed immissione dei caratteri in memoria (linea 90). I dati letti e memorizzati sono in tutto 72 (0-71), perchè abbiamo 9 righe di 8 byte ciascuna. Per aggiungere delle linee, modificate il FOR della linea 70 in modo che conti da 12288 a 12288+(n – 1), dove n è il numero

complessivo dei byte da leggere. Ad esempio, per leggere 160 caratteri (20 linee DATA di 8 byte ciascuna), usate una linea come:

```
70 FOR J=12288 TO 12288+(160-1)
```

MODO CARATTERE MULTICOLORE

Il modo carattere standard mostra il testo in 2 colori: il colore di primo piano, determinato per ogni singolo carattere dalla RAM colore, ed il colore di fondo, unico per tutti i caratteri, determinato dal registro 0 in 53281.

Il modo carattere multicolore vi dà la possibilità di usare 4 colori entro una matrice carattere di 8x8 pixel. Questo in sostanza aumenta la libertà d'uso dei colori, a prezzo però della risoluzione orizzontale, che è metà del normale. Ciò significa che la definizione del colore e la densità dei pixel sono doppie rispetto al normale. La perdita di risoluzione orizzontale è compensata dalla maggior libertà d'uso dei colori.

COME ENTRARE IN MODO MULTICOLORE

La locazione 53270 ed il suo registro ombra 216 determinano se il C128 visualizza caratteri standard o multicolore sullo schermo. Il bit 4 di 53270 ed il 7 di 216 controllano il multicolore, sia in modo testo che bit map. Se, con l'editor grafico attivo, il bit 7 di 216 vale 1, allora entrate in bit map multicolore; viceversa siete in modo testo standard. Molti dei nuovi comandi grafici del BASIC 7.0

prevedono l'uso del multicolore. Se volete comunque usare le istruzioni POKE:

POKE 216,255: REM disabilita l'editor di schermo

POKE 53270,PEEK(53270) OR 16: REM abilita il modo multicolore

Ciò abilita il multicolor sia in modo testo che bit map.

LOCAZIONI DI SCHERMO

Le locazioni della memoria di schermo multicolore sono le stesse del modo standard, da 1024 a 2023, e possono essere rilocate. Per i dettagli andate alla sezione della Memoria di Schermo.

INTERPRETAZIONE DEI DATI DELLO SCHERMO

In modo multicolore, i dati contenuti nella memoria di schermo sono sempre interpretati come normali codici di schermo. I codici di schermo sono elencati nel Volume 8, Appendice D. La sola differenza dal modo standard è l'assegnazione del colore ai singoli caratteri.

LOCAZIONI DELLA MEMORIA CARATTERI

Anche in modo multicolore la memoria caratteri è normalmente locata a 53248.

INTERPRETAZIONE DELLA MEMORIA CARATTERI IN MODO CARATTERE MULTICOLORE

La memoria carattere, sia in modo standard che multicolore, è virtualmente interpretata allo stesso modo, eccetto una cosa: in modo standard, se un bit del carattere vale 1, il pixel corrispondente sarà del colore di primo piano; viceversa, il pixel sarà del colore di fondo.

In modo multicolore, la relazione fra pixel e colore non è così diretta.

Invece, i bit che compongono un carattere vengono presi a coppie, come mostrato nelle Figure 9-18, 9-19 e 9-20.

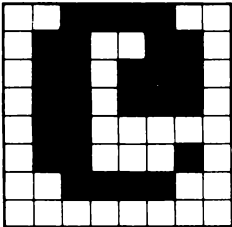



Figura 9-18. Il carattere @ come appare sullo schermo

0	0	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	1	1	1	0
0	1	1	0	1	1	1	0
0	1	1	0	0	0	0	0
0	1	1	0	0	0	1	0
0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0

Figura 9-19. Maschere dei bit nella ROM caratteri

00	11	11	00
01	10	01	10
01	10	11	10
01	10	11	10
01	10	00	00
01	10	00	10
00	11	11	00
00	00	00	00

Figura 9-20. Maschere dei bit presi a coppie in modo multicolore

I bit sono raggruppati a coppie, da qui il dimezzamento della risoluzione orizzontale. La coppia di bit determina il colore del pixel a cui corrisponde. La sezione seguente descrive l'assegnazione dei colori.

DATI DEI COLORI

I colori dei pixel in modo multicolore provengono da 4 sorgenti, a seconda della coppia di bit. Poichè ogni coppia può scegliere fra 4 colori, occorrono 2 bit per scegliere la sorgente: 00, 01, 10 e 11. In Figura 9-21 sono descritte le sorgenti corrispondenti ad ogni combinazione di bit:

COPPIA	REGISTRO COLORE	LOCAZIONE
00	Fondo #0	53281 ($D021)
01	Fondo #1	53282 ($D022)
10	Fondo #2	53283 ($D023)
11	I 3 bit più bassi della RAM colore	

Figura 9-21. Tabella delle corrispondenze dei colori

Dunque, se la coppia di bit è 00, il pixel sarà del colore in 53281, se 01 del colore in 53282, se 10 di quello in 53283 ed infine se 11 del colore specificato dai 3 bit più bassi (2, 1 e 0) della RAM colore. Quest'ultima é locata da 55296 a 56295.

In modo carattere multicolore, potete visualizzare ancora caratteri in modo standard. Infatti è possibile scegliere il modo per ogni singolo carattere, impostando il bit 3 della RAM colore. Se questo bit vale 0, il modo del carattere associato è standard, altrimenti multicolore. Ciò significa che in modo multicolore potete usare solo i primi 8 colori, qualunque sia lo stato del bit 3. Se volete dunque visualizzare un carattere in bianco standard, premete CTRL contemporaneamente al tasto 2; se lo volete in bianco multicolore (oltre ad altri tre colori) premete il tasto **Commodore** e 2. La Figura 9-22 mostra i colori disponibili:

CODICE	COLORE
0	Nero
1	Bianco
2	Rosso
3	Azzurrino
4	Porpora
5	Verde
6	Blu
7	Giallo

Figura 9-22. Colori disponibili in modo multicolore

Ricordate che il bit 3 della RAM colore deve essere a 1 per visualizzare un carattere multicolore. Il programma seguente illustra il modo carattere multicolore:

```
10 COLOR0,1:REM FONDO NERO
20 COLOR2,2:REM MULTI1 BIANCO
30 COLOR3,3:REM MULTI2 ROSSO
40 FORI=1TO25
50 PRINT"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
60 NEXT
70 FORI=55296+512TO55296+1023:POKEI,7:NEXT:REM GIALLO NELLA RAM COLORE
80 POKE216,255:REM DISABILITA L'EDITOR GRAFICO
90 POKE53270,PEEK(53270)OR16:REM IMPOSTA IL MULTICOLORE
100 FORI=1TO25
110 PRINT"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
120 NEXT
130 FORI=55296TO55296+1023:POKEI,14:NEXT:REM BLU' NELLA RAM COLORE
140 GRAPHIC0:REM RIATTIVA L'EDITOR GRAFICO
```

Le linee 10, 20 e 30 pongono i colori nero, bianco e rosso rispettivamente nei registri 0, 1 e 2. Le linee da 40 a 60 stampano 25 volte le lettere dell'alfabeto. La linea 70 riempie gli ultimi 512 byte della RAM colore col codice 7 (giallo). La linea 80 disabilita l'editor grafico. La 90 abilita il modo multicolore. A questo punto, tutte le locazioni di schermo che hanno un colore superiore a 7 saranno in modo multicolore. Poichè il giallo ha codice 7, tutti i caratteri corrispondenti saranno in modo standard. Il colore di default della RAM colore è 13 per il C128 e 14 per il C64. Le linee 100, 110 e 120 riscrivono 25 volte le lettere dell'alfabeto. La linea 130 riempie la RAM colore col codice 14. I caratteri saranno così rossi, bianchi e blú su sfondo nero.

MODO COLORE DI FONDO ESTESO

Il terzo tipo di visualizzazione, il modo colore di fondo esteso, vi permette di scegliere tra 4 colori di fondo per ogni carattere. Per esempio, potete avere il colore del carattere, il suo colore di fondo e in tutto il resto dello schermo il normale colore di fondo. Ad esempio, questo significa che è possibile avere un carattere bianco su un suo fondo verde col fondo restante nero. Questo modo offre un colore extra per ogni carattere, senza alcuna perdita di risoluzione.

C'è comunque un sacrificio. In modo colore di fondo esteso sono usabili solo i primi 64 caratteri di un set, perdendo gli altri 192. Ciò si spiega col fatto che 2 bit del codice carattere sono usati per specificare 1 fra 4 colori prescelti: i bit restanti sono 6, quindi $2^6=64$.

COME ENTRARE IN MODO COLORE DI FONDO ESTESO

Ponete a 1 il bit 6 di 53265 per selezionare il colore di fondo esteso. Fate così in BASIC:

POKE 53265, PEEK(53265) OR 64

Per uscire dal modo:

POKE 53265, PEEK(53265) AND 191

LOCAZIONI DI SCHERMO

Le locazioni di schermo sono le stesse dei modi standard e multicolore (1024-2023), e possono essere spostate in memoria. Ciò é spiegato in dettaglio nella sezione della Memoria di Schermo.

INTERPRETAZIONE DEI DATI DI SCHERMO

I dati nella memoria di schermo sono letti come codici di schermo, cioè in realtà dei puntatori alla ROM. Quindi non corrispondono ai codici ASCII, ma seguono l'ordine di memorizzazione nella ROM. Quindi, essendo @ il primo carattere in ROM, il suo codice sarà 0.

Ricordate che solo i primi 64 caratteri sono disponibili, perchè i 2 bit rimanenti selezionano il colore di fondo per quel carattere.

DATI DEI COLORI

I colori di una posizione carattere provengono da 3 fonti diverse. Il colore di primo piano è dato dalla RAM colore (55296–56295), che ha una corrispondenza univoca fra ogni locazione di schermo e la corrispondente nella RAM colore. Le mappe delle due memorie si trovano nella sezione del Modo Carattere Standard assieme alla relazione fra le due zone di memoria.

Il colore di fondo, per ogni singolo carattere, proviene invece da 1 fra 4 registri del colore di fondo. I bit 6 e 7 del codice di schermo formano infatti 4 combinazioni. La Figura 9-23 mostra la relazione fra la combinazione di bit e la sorgente di colore.

CODICE CARATTERE			COLORE DI FONDO	
CAMPO	**BIT 6**	**BIT 7**	**NUM.**	**INDIRIZZO**
0–63	0	0	0	53281
64–127	0	1	1	53282
128–191	1	0	2	53283
192–255	1	1	3	53284

Figura 9-23. Registri del colore di fondo esteso

Per esempio, inserite in 1024 con POKE il codice della lettera A (1). Ora inserite in 1025 il valore 129. Potreste aspettarvi una A in negativo, visto che ha un codice di schermo maggiore di 127. Invece non avete fatto altro che porre a 1 il bit 7 del codice di schermo. Vedrete così una A dello stesso colore della prima, ma su un fondo preso dal registro in 53283 (registro di fondo #2). Un metodo

semplice per ottenere i caratteri scegliendo fra i 4 colori di fondo consiste nel digitarli da tastiera assieme a certi tasti speciali, che influenzano i bit 6 e 7. Questi tasti sono:

- nessuno per caratteri con fondo da 53281
- SHIFT per il colore di fondo da 53282
- CTRL+9 e successivamente le due combinazioni sopra per avere rispettivamente i colori da 53283 e 53284.

La spiegazione di questo funzionamento è semplice: un carattere stampato in modo normale senza SHIFT ha un codice di schermo che arriva fino a 63, cioè coi bit 6 e 7 a 0. Con lo SHIFT, si aggiunge 64 al codice, mettendo così a 1 il bit 6. Passando in reverse con CTRL+9 non fate altro che aggiungere 128 ai caratteri stampati in seguito, ponendo così a 1 il bit 7. Con lo SHIFT aggiungete, oltre a 128, anche 64, mettendo a 1 entrambi i bit 6 e 7. Per uscire dal reverse usate CTRL+0. Combinando correttamente questi tasti potete comporre delle stringhe con sequenze di caratteri dal fondo diverso, senza usare POKE. Ecco un programma dimostrativo del colore di fondo esteso:

```
10 SCNCLR
20 COLOR0,1:REM FONDO NERO
30 COLOR2,2:REM MULTI1 BIANCO
40 COLOR3,3:REM MULTI2 ROSSO
50 POKE53284,3:REM FONDO #3 IN AZZURRINO
60 POKE53265,PEEK(53265)OR64:REM IMPOSTA IL MODO COLORE DI FONDO ESTESO
70 FORI=1024TO1279:POKEI,I-1024:NEXT
80 CHAR,0,6,""
```

Nel programma, la linea 10 pulisce lo schermo. Le linee da 20 a 50 assegnano il colore ai 4 registri di fondo: nero, bianco, rosso e azzurrino. La linea 60 abilita il modo colore di fondo esteso. La linea 70 inserisce 256 caratteri nella memoria di schermo.

I codici di schermo variano così da 0 a 63, da 64 a 127, da 128 a 191 e da 192 a 255, prendendo così un diverso colore di fondo per ogni 64 caratteri.

DATI DEI CARATTERI

Questi dati sono locati nelle stesse posizioni degli altri modi carattere; vengono però usati solo i primi 64 caratteri.

MODO BIT MAP STANDARD

Il modo bit map standard, spesso chiamato modo ad alta risoluzione, dà la possibilità di visualizzare immagini grafiche dettagliate in 2 colori. La risoluzione è di 320x200 pixel. In questo modo, il C128 non ragiona più in termini di caratteri, che sono matrici di 8x8 pixel, ma vi permette di indirizzare ogni singolo pixel, aumentando così il controllo del video. Se quindi nel modo carattere potevate controllare solo 8x8 pixel alla volta, qui sceglierete 1 su 64000 pixel. La Figura 9-24 mostra il piano di coordinate bit map:

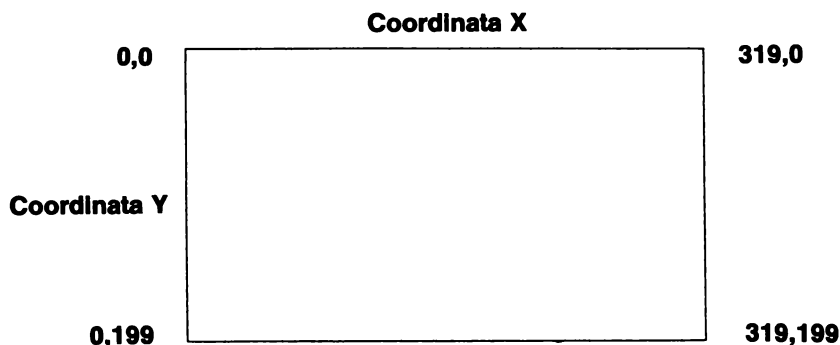



Figura 9-24. Coordinate dello schermo bit map

COME ENTRARE IN MODO BIT MAP STANDARD

Per entrare in bit map standard, ponete a 1 il bit 5 di 216 (il registro ombra di 53265). Lo stesso effetto si ottiene col comando GRAPHIC 1,1, ottenendo in più la cancellazione dello schermo.

Se preferite non fare uso dei comandi ad alto livello, usate questa POKE:

POKE 216, PEEK(216) OR 32

Questo comando mette a 1 il bit 5 del registro GRAPHM, che fa da interfaccia fra il chip VIC e l'editor grafico del C128. Cosí mettete a 1 indirettamente il bit 5 di 53265.

Potete disabilitare l'editor grafico e selezionare direttamente il modo bit map con questi comandi:

POKE 216, 255 POKE 53265, PEEK(53265) OR 32

In linguaggio macchina, sul C128, usate questa sequenza:

```
LDA $D8
ORA #$20
STA $D8
```

In linguaggio macchina, sul C64, provate questo:

```
LDA $D011
ORA #$20
STA $D011
```

LE LOCAZIONI DELLA MATRICE VIDEO (MEMORIA DI SCHERMO)

Sul C128, queste locazioni vanno da 7168 ($1C00) a 8167 ($1FE7).

Sul C64 vanno invece da 1024 ($0400) a 2023 ($07E7). In entrambi i casi la matrice video può essere rilocata. Ciò viene spiegato nella sezione della Memoria di Schermo, altrove in questo capitolo.

INTERPRETAZIONE DELLA MATRICE VIDEO

In modo bit map, la matrice video ha un significato diverso rispetto al modo carattere. Quest'ultimo infatti interpreta ogni byte dello schermo come un puntatore alla ROM caratteri. Per il modo bit map invece ogni byte fornisce il colore di una zona di pixel: i 4 bit alti il colore di primo piano, i 4 bit bassi il colore di fondo.

La sezione seguente spiega l'assegnazione dei colori ai pixel, sia per fondo che primo piano.

DATI DELLA BIT MAP

In modo bit map i dati sui caratteri non sono presi dalla ROM, ma da una zona di 8K di RAM conosciuta appunto come bit map.

Lo schermo ad alta risoluzione standard è formato da 200 linee di 320 pixel l'una, così lo schermo intero è composto da 64000 pixel. Ogni pixel è legato ad un bit di memoria, così sono necessari 64000 bit (o 8000 byte) di RAM per l'intera immagine.

Un bit a 1 produrrà un pixel acceso, cioè del colore fornito dai 4 bit alti della matrice video, mentre logicamente un bit 0 corrisponderà ad un pixel spento, del colore dei 4 bit bassi. L'insieme di pixel accesi e spenti forma l'immagine ad alta definizione.

Le locazioni di default della bit map vanno da 8192 ($2000) a 16191 ($3F3F). I byte da 16192 a 16383 sono usati per altri scopi. La locazione 53272 determina le locazioni della bit map e della matrice video. Voi dovete però modificare la locazione 2605, usata dall'editor di schermo per modificare 53272. I 4 bit alti di 2605 determinano la locazione della matrice video, mentre i 4 bassi l'inizio della bit map. Dei 4 bassi, solo il bit 3 è significativo per la locazione, poichè in un banco video si hanno solo 2 locazioni possibili, da 0 o da 8192.

Se passate ad un banco video superiore allo 0, ricordate di aggiungere lo spostamento di $4000 per banco.

Per la rilocazione della matrice video potete riferirvi alla sezione della Memoria di Schermo, all'inizio di questo capitolo.

Potete pensare alla bit map come ad un pannello luminoso composto da tanti led. Accendendoli opportunamente, si formano scritte e disegni. Per esempio, se la bit map iniziasse a 8192 (default in BASIC C128), il primo byte conterrebbe le

informazioni dei pixel dalle coordinate 0,0 a 7,0. Il secondo byte, in 8193, contiene i pixel di coordinate da 0,1a 7,1. La Figura 9-25 mostra la disposizione dei byte della bit map rispetto ai pixel di schermo:

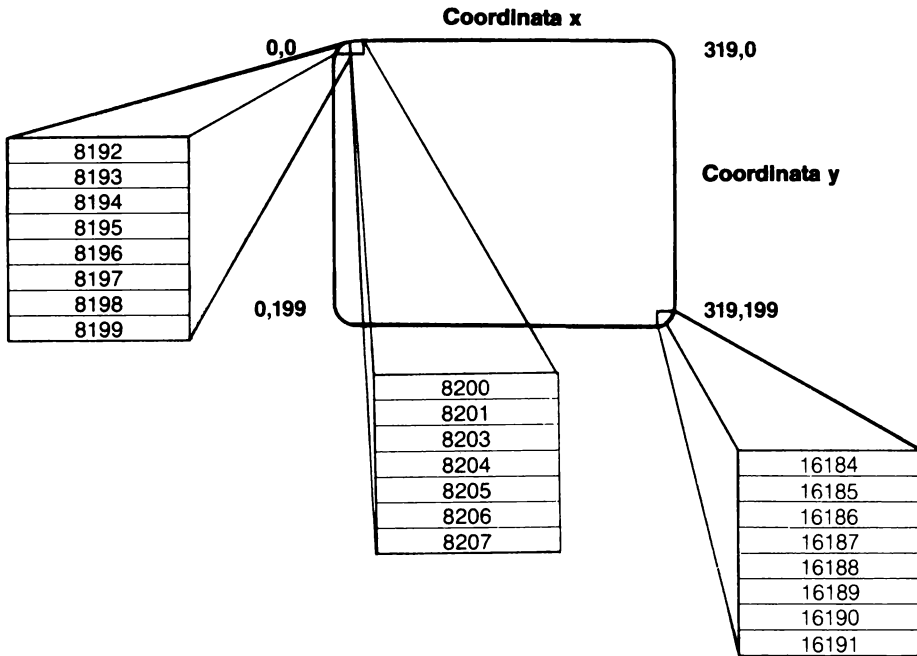



Figura 9-25. Disposizione di byte e pixel della bit map

Due sono i modi principalmente usati per calcolare, date le coordinate, l'indirizzo di un byte della bit map. Il primo metodo consiste nell'usare la seguente formula:

(YAND248)*40+(XAND248)+(YAND7)

con X e Y varianti rispettivamente da 0-320 e da 0-199. Valori maggiori sconfineranno oltre la zona di 8K prescelta.
La logica di funzionamento sta nei seguenti passi:

- immaginiamo di raggruppare le 200 linee verticali a gruppi di 8, ottenendo così 25 zone di 320 byte ciascuna (40 byte x 8 linee);

- dividiamo per 8 la Y, ottenendo così una parte intera che indica il numero di zone da 320 byte ed un'eventuale parte decimale che lasceremo da parte;

- ricaviamo la prima parte della formula: INT(Y/8)*320, che ci dà un primo indirizzo. Possiamo effettuare una semplificazione: (YAND248)*40. Infatti la parte decimale del quoziente proviene dai 3 bit bassi (0-7), che vengono azzerati dall'AND248. A questo punto avremmo (YAND248)/8*320, cioè (YAND248)/40;

- abbiamo l'indirizzo del primo byte della zona finale puntata dalla Y. Aggiungiamo la parte della X multipla di 8 (XAND248) ed arriviamo al primo degli 8 byte formanti una casella, come visto da Figura 9-25;

- sommiamo ora la parte bassa della Y (YAND7), che è lo spostamento all'interno della casella. Adesso abbiamo l'indirizzo del byte in cui cade il pixel di coordinate X,Y;

- per accedere al singolo pixel utilizziamo i 3 bit bassi della X, che danno lo spostamento all'interno del byte (0-7);

- ora, usando questi 3 bit come un indice, effettuiamo un'operazione logica (AND, OR, EOR) per operare sul pixel.

Il secondo metodo è più diretto ed ha una velocità quasi doppia: consiste nell'usare una tabella contenente gli indirizzi iniziali di ogni zona da 320 byte. Ad ogni indirizzo si accede dividendo per 8 la Y ed usando il risultato come un indice nella tabella. In pratica viene eliminata la prima operazione della formula sopra descritta ((YAND248)*40).
La costruzione della tabella è molto semplice: partite da 8192, che sarà il primo indirizzo, aggiungete 320 e troverete il secondo e così via fino alla 24esima zona.
Entrambi i metodi sono ottimi sia per il BASIC che specialmente per il linguaggio macchina. Diciamo che un algoritmo impiegante il primo metodo richiederà come minimo 87 cicli di clock, mentre con una tabella il tempo si ridurrà a 50 cicli.

Ora conoscete il funzionamento del modo bit map del C128. Se non volete o non avete la capacità di scrivere le vostre proprie routine potete usare i nuovi comandi ad alto livello del BASIC 7.0, come DRAW, CIRCLE, BOX e PAINT.

L'uso di un sistema di coordinate renderà più facile la visualizzazione di grafici. Al di fuori dei comandi grafici del BASIC 7.0 vi sono altri modi di tracciare punti in alta risoluzione. Ciò comprende programmi commerciali impieganti tavolette grafiche o joystick, oppure l'immissione diretta dei dati, con POKE, nella memoria grafica (cosa sconsigliabile).

Un'altra maniera di disegnare dei grafici è tramite la geometria, usando formule ed equazioni matematiche. Sono disponibili parecchi libri sul disegno ed il movimento di oggetti tridimensionali.

RAM COLORE

In modo bit map standard, la RAM colore non é usata, poiché ci si basa sui semibyte della matrice video. Se ne farà uso nel modo bit map multicolore.

MODO BIT MAP MULTICOLORE

Il modo bit map multicolore è una combinazione della bit map standard e del modo carattere multicolore. Questo modo vi permette di usare fino a 4 colori in un'area di 8x8 pixel. Così come nel modo carattere multicolore, la risoluzione orizzontale si dimezza, sebbene questa perdita sia compensata dal maggior uso del colore.

COME ENTRARE IN MODO BIT MAP MULTICOLORE

Per entrare in bit map multicolore da BASIC, fate:

GRAPHIC 3

Potete anche usare un comando POKE, nonostante sia consigliabile usare sempre i comandi ad alto livello:

POKE 216, PEEK(216) OR 160

Il valore 160 (128+32) mette a 1 i bit 7 e 5 di 216 (il registro ombra usato dall'editor grafico) ed indirettamente il bit 5 di 53265 (modo bit map) ed il bit 4 di 53270 (modo multicolore).

Quando il bit 5 di 53265 vale 1, si è in modo bit map; così per il bit 4 di 53270, che se è a 1 visualizza in multicolore sia testo che bit map. In modo 64, potete scrivere direttamente in 53265 e 53270. In modo 128, a causa dell'editor grafico, dovete per forza usare i registri ombra. Ciò rende possibili le schermate divise, in parte bit map ed in parte testo.

Il registro GRAPHM ($D8) è quello più usato, i cui contenuti vengono usati per impostare 53265, 53270 e 53272 ogni 1/60 di secondo.

LINGUAGGIO MACCHINA

Per passare in modo multicolore da linguaggio macchina, usate questa sequenza:

```
LDA #$A0      imposta la bit map multicolore
STA $D8
```

Sul C64, invece:

```
LDA $D011
ORA #$20      modo bit map
STA $D011
```

```
LDA $D016
ORA #$10     modo multicolore
STA $D016
```

In entrambi i casi, il vostro programma dovrà poi cancellare lo schermo e la memoria colore.

LOCAZIONI DELLA MATRICE VIDEO

La matrice video è locata di default da 7168 ($1C00) a 8167 ($1FE7) in modo 128.

In modo 64 si trova invece da 1024 ($0400) a 2023 ($07E7). Entrambe sono rilocabili. A ciò andate alla sezione della Memoria di Schermo più indietro nel capitolo.

INTERPRETAZIONE DELLA MATRICE VIDEO

La matrice video è interpretata sempre a semibyte (nybble), come nel modo bit map standard.

Il nybble alto fornisce il colore di una coppia di bit, mentre quello basso il colore di un'altra.

DATI COLORE ADDIZIONALI

I due colori addizionali dei 4 offerti dal modo bit map multicolore provengono dal semibyte basso della RAM colore e dal registro 0 del colore di fondo in 53281.

Come in modo carattere multicolore, le coppie di bit determinano il colore del pixel ad esse associato, secondo la seguente tabella:

BIT	FONTE DEL COLORE
00	Registro 0 di fondo (53281)
01	Semibyte alto della matrice video
10	Semibyte basso della matrice video
11	RAM colore

Figura 9-26. Fonti di colore in modo bit map multicolore

LA BIT MAP

Le coppie di bit determinano la provenienza del colore. Una coppia 00 avrà il colore di fondo (53281); una 01 il colore del semibyte alto della matrice video; una 10 quello del semibyte basso ed infine una 11 quello della RAM colore. Al contrario del modo carattere, non è possibile selezionare fra parti in risoluzione standard e parti in multicolore, a meno che non sviluppiate un complesso programma di suddivisione dello schermo in due parti tramite interruzione raster.

RAM COLORE

In modo bit map multicolore, la RAM colore è usata dalle coppie di bit 11. Ogni locazione della RAM colore può avere fino a 16 colori, il che significa che ogni area di 8x8 pixel può essere dei colori nerô, rosso, bianco e blu presi rispettivamente dal registro di fondo 0, dai semibyte alti e bassi della matrice video e dalla RAM colore. L'area di 8x8 pixel adiacente può avere il colore nero più altri 3 del tutto differenti, poichè presi da un'altra casella della matrice video e della RAM colore. Il colore delle coppie 00 resta lo stesso per tutto lo schermo.

Il C128 ha varie e potenti capacità grafiche. Alcune applicazioni useranno un certo tipo di schermo piuttosto che un altro. Dovreste sperimentare coi vari modi per vedere quello che meglio si adatta alle vostre esigenze. Le Figure da 9-28 a 9-32 forniscono un sunto della programmazione grafica utile per capire la grafica del C128.

MODI A SCHERMO DIVISO

Il Commodore 128 ha la caratteristica di poter visualizzare la parte superiore dello schermo in modo bit map, e la parte inferiore in modo testo. Questo vi permette di inserire ed eseguire un programma grafico in BASIC vedendone contemporaneamente i risultati ed il listato, ed evitandovi così di perdere tempo a passare da testo a grafica e viceversa.

Prima, avreste dovuto inserire il vostro programma (in linguaggio macchina), passare in grafica ed eseguirlo, poi tornare in modo testo per apportare delle modifiche. Ora, potete visualizzare l'immagine grafica e avere allo stesso tempo disponibile lo schermo di testo. Potete modificare il programma col disegno sempre sullo schermo, eseguirlo e vederne i risultati senza perdere lo schermo di testo.

Senza la capacità del Commodore 128 di dividere lo schermo, avreste dovuto scrivere da voi stessi un programma che lo facesse. Questo implica interruzioni raster che utilizzino due memorie di schermo ognuna in due banchi video differenti, oppure una singola memoria di schermo abbastanza perturbata e

solitamente con una linea di divisione visibile. Con il modo a schermo diviso del C128, tutto quello che avete da fare è dare il comando GRAPHIC da BASIC. Per esempio, il comando:

GRAPHIC 2,1

imposta una bit map standard con 5 linee di testo in basso. Similmente, il comando:

GRAPHIC 4,1

costruisce uno schermo bit map multicolore nella parte alta dello schermo, lasciando la parte bassa in modo testo. L'1 in questi comandi fa cancellare lo schermo. Per evitare la cancellazione, ponete uno 0 al posto dell'1 od omettetelo del tutto.

Il comando GRAPHIC ha un parametro addizionale che vi permette di definire il punto di divisione dello schermo. Questo punto si definisce come riga da 0 a 24, e di default si pone a 20. Per esempio:

GRAPHIC 4,1,15

seleziona uno schermo diviso con bit map multicolore e 10 righe di testo (da 15 compresa a 24).

ORGANIZZAZIONE IN MEMORIA DEI MODI A SCHERMO DIVISO

LOCAZIONI DI SCHERMO

I modi a schermo diviso, sia standard che multicolore, usano due distinte memorie di schermo. La matrice video per la bit map proviene da 7168 ($1C00) a 8191 ($1FFF), come nei modi bit map a schermo intero. La porzione di testo usa invece la memoria da 1024 ($0400) a 2023 ($07E7), come nei modi di testo. Le porzioni di schermo nascoste contengono lo stesso dei dati, ma non sono visibili.

INTERPRETAZIONE DEI DATI DI SCHERMO

Le porzioni di testo e bit map sono viste esattamente come nei modi normali. Consultate le sezioni del Modo Carattere Standard e dei Modi Bit Map Standard e Multicolore per i dettagli.

LOCAZIONI DELLA MEMORIA CARATTERE

Gli schermi divisi prendono le informazioni sui caratteri da due diverse zone di memoria. I dati bit map provengono dalla zona da 8192 ($2000) a 16191 ($3F3F), sia in modo standard che multicolore.

La memoria carattere per la parte di testo proviene invece dalla ROM caratteri. La vera ROM occupa gli indirizzi da 53248 ($D000) a 57343 ($DFFF), sotto l'I/O. Per esigenze del chip VIC, compare invece un'immagine della ROM da 4096 a 8191: è questa immagine che viene resa visibile per la parte di testo.

Per informazioni sull'interpretazione dei dati carattere nei modi testo e bit map, andate alle sezioni relative. Date anche un'occhiata alla parte di suddivisione a banchi della RAM colore.

DATI COLORE

Ognuno dei modi usati interpreta diversamente la RAM colore. Riferitevi ad ogni sezione per i dettagli.

LINGUAGGIO MACCHINA

In linguaggio macchina, dovete fornire voi stessi il programma che divide lo schermo. Non è dei più semplici compiti di programmazione, poichè implica la gestione dell'interruzione raster, cosa che può essere rischiosa. In modo 128, il bit 6 di GRAPHM ($D8) controlla la divisione dello schermo. Mettetelo a 1 per far

dividere lo schermo dall'editor grafico. Altrimenti avrete uno schermo completamente bit map.

Il modo 64 non ha un bit corrispondente, e richiede un programma esterno. Esaminate il Programma di Divisione dello Schermo a Raster Interrupt alla fine del capitolo per imparare la tecnica di divisione dello schermo da linguaggio macchina.

> **ATTENZIONE:** Può succedere un blocco del sistema se effettuate un cambiamento di modo video con l'editor grafico attivo. Andate alla sezione dei Registri Ombra per i dettagli.

L'EDITOR DI SCHERMO AD INTERRUZIONE

Le locazioni intermedie di memoria, spesso chiamate registri ombra, sono incluse apposta per i modi a schermo diviso. Allo scopo di fornire i modi divisi, l'editor di schermo del C128 ha dovuto entrare a far parte delle routine di gestione delle interruzioni.

Al contrario del C64, il C128 maneggia le interruzioni basandosi esclusivamente sul raggio del raster. Questo si è reso necessario per inserire l'editor del 128 entro le routine d'interruzione. Il C64 usa dei timer d'interruzione che rendono meno prevedibile il processo di gestione delle interruzioni. Invece, agganciandosi al raggio del raster, il sistema operativo sa sempre dove e quando succederà un'interruzione.

Gli interrupt da timer rendono meno sicura la risposta ad un raster interrupt perchè il sistema operativo non sa mai esattamente quando avverrà l'interruzione da raster. L'editor di schermo ad interruzioni rende necessario l'uso di locazioni indirette per l'aggiornamento di alcuni registri dei chip VIC e 8563 (80 colonne). In questo modo ad ogni nuova scansione dello schermo, ogni 1/60 di secondo, i veri registri video vengono caricati dalle locazioni indirette.

PROGRAMMA A SCHERMO DIVISO TRAMITE INTERRUZIONI, CON SCROLL ORIZZONTALE

Questa sezione fornisce e spiega un programma in linguaggio macchina per gestire uno schermo diviso ad interruzioni. Questo programma gira in modo 64, ma può essere adattato al modo 128.

In modo 128 avete già la possibilità di dividere lo schermo col comando GRAPHIC. Il programma qui fornito divide lo schermo in modo 64. Osservate la figura nella sezione dei Registri Ombra per vedere gli indirizzi da modificare nell'adattamento al modo 128. Ci sarà una lieve differenza nella temporizzazione del raster. In modo 128, tutte le interruzioni avvengono secondo la posizione del pennello elettronico sullo schermo. Questa è la ragione per cui esistono i registri ombra per alcune locazioni grafiche, in modo da distinguere fra i vari modi attivi.

Il programma in questa sezione esegue lo scroll del testo nel quarto inferiore dello schermo, mentre i tre quarti superiori sono in modo bit map multicolore. Lo schermo di testo standard risiede nel banco video 0 ($0400–$07E7), mentre la bit map multicolore è memorizzata nel banco 1 a partire da $5C00 ($1C00+4000=$5C00). Ad ogni occorrenza di un'interruzione, il programma cambia banco video, modo grafico, locazioni delle memorie caratteri e video. Il programma fornisce i dati di testo da far scrollare, ma assume che abbiate posto una pagina grafica a partire da 24576 ($6000) con la sua memoria colore da 23552 ($5C00), e cioè nel banco video 1.

Una bit map di 8000 byte é leggermente più grande di quanto consentito da questo libro. Comunque, ecco un modo facile di produrre una pagina grafica:

1. Entrate in modo 128 e date il seguente comando:

   GRAPHIC 4,1

2. Ora disegnate sullo schermo coi comandi BOX, CIRCLE, DRAW e PAINT, sia da programma che direttamente.

3. Quando avete finito di disegnare, entrate in monitor premendo F8 o dando il comando MONITOR.

4. Adesso trasferite la matrice video e la bit map dalle locazioni di default sul C128, rispettivamente da $1C00 a $1FFF e da $2000 a $3FFF, a partire da $5C00 a $5FFF e da $6000 a $7FFF. I nuovi indirizzi di partenza sono le locazioni di default più uno spostamento di $4000 sia per la matrice video che per la bit map. La nuova matrice video parte a $5C00 ($1C00+$4000). La nuova bit map parte a 6000 ($2000+$4000). Questo trasferimento può essere eseguito da monitor col comando seguente:

   T 1C00 3FFF 5C00

Questo comando trasferisce i contenuti delle locazioni da $1C00 a $3FFF da $5C00 a $7FFF. Il trasferimento può essere fatto con un solo comando poichè le locazioni della matrice video e della bit map sono contigue. Il Capitolo 6 spiega l'uso del Monitor.

Ora che siete ancora in monitor, cominciate a ricopiare il disassemblato che appare nelle prossime pagine. Iniziate dall'indirizzo $0C00. Il programma, inclusi i dati di testo, occupa la memoria fino a $0DF9, che significa un lunghezza di 505 byte, quasi mezzo Kbyte.

Ora salvate il programma che avete diligentemente ricopiato, usando il comando Save (S) del monitor come segue:

S"nome del file", 8, 0C00, 0DFF

Se avete un assemblatore per C128, ricopiate invece il listato sorgente (quello con etichette e commenti), assemblatelo e caricatelo e salvatelo come file binario.

> **NOTA:** Se avete il Sistema di Sviluppo Assembler del Commodore 64, ricopiate sempre il listato sorgente (con indirizzo di partenza $0C00), assemblatelo e caricatelo in modo 64. Poi premete il pulsante di RESET (non spegnete la macchina!) per entrare in modo 128. Non temete, il vostro programma è ancora in memoria, solo che adesso è nella memoria del C128. A questo punto entrate in monitor e usate il comando S visto sopra.

Ora siete pronti ad andare in modo 64:

GO 64

Rispondete alla domanda ARE YOU SURE? (sei sicuro?) con Y e premete <RETURN>. Ora siete in modo 64.

A questo punto, potreste chiedervi: "Dopo che ho fatto tutto questo lavoro, chi me lo fa fare di distruggerlo cambiando di modo?".

In realtà, non state distruggendo nessuno dei vostri sforzi. Quando andate in modo 64 (o premete il pulsante di RESET), molta della RAM dei programmi in linguaggio macchina e dei dati viene conservata nel banco RAM 0. Solo i programmi BASIC vengono cancellati. Ecco le parti di RAM lasciate inalterate nel passaggio dal modo 128 al modo 64:

DISPOSIZIONE DI MEMORIA DEL C128

$0C00-0DFF	**Buffer di I/O dell'RS232**
$1300-1BFF	**Memoria libera per applicazioni varie**
$1C00-1FFF	**Matrice video per la bit map**
$2000-3FFF	**Dati della bit map**
$4000-FF00	**Area dei programmi BASIC**

Figura 9-27. RAM inalterata dal modo 128 al modo 64

Il resto della RAM è usato per altri scopi e cambia da modo a modo. Ci sono inoltre altri byte particolari che rimangono inalterati, ma non vale la pena di menzionarli. I blocchi di memoria sopra menzionati forniscono un'indicazione generale della RAM usata da entrambi i modi. Ricordate comunque che i contenuti della RAM, di qualunque locazione, andranno perduti se spegnete e riaccendete il computer invece di premere il tasto di RESET.

Notate che gli indirizzi in cui ponete il vostro programma, la matrice video e la bit map sono fra quelli lasciati inalterati.

Ora fate partire il programma, dal modo 64, con questo comando:

SYS 12*256

I tre quarti superiori dello schermo saranno in modo bit map e vedrete i disegni da voi creati, mentre il quarto inferiore sarà in modo testo con lo scroll dei caratteri.

I programmi seguenti sono rispettivamente il listato sorgente (per chi ha un assemblatore) ed il disassemblato (per chi usa il monitor) per dividere lo schermo e fare lo scroll del testo.

```
1000 ; SCHERMO DIVISO TRAMITE INTERRUPT CON SCROLL ORIZZONTALE
1010 IVEC   = $0314
1020 TEMP   = $1806
1030 RASTRO = $D012
1040 BACOL  = $D021
1050 POINT  = $1802
1060 FLAG   = $FC
1070 FLAG2  = $FD
1080 SCROLL = $FE
1090 SREG   = $D016
1100 TXTPTR = $FA            ; PUNTATORE DELLO SCROLL DEL TESTO
1110 ;
1120 *=$0C00
1130 ;
1140        LDA #<FRANK      ; PUNTA AL TESTO
1150        STA TXTPTR
1160        LDA #>FRANK
1170        STA TXTPTR+1
1180 ;
1190        LDA #7           ; VALORE MASSIMO DEL REGISTRO DI SCROLL
1200        STA SCROLL
1210        STA FLAG2
1220 ;
1230        LDA $DD02        ; REGISTRO IN USCITA
1240        ORA #3
1250        STA $DD02
1260 ;
1270 ;
1280        LDA $D016        ; 38 COLONNE
1290        AND #$F7
1300        STA $D016
1310 ;
1320 ;
1330        LDA #1           ; PULISCE LA RAM COLORE
1340        LDX #0
1350 BONK   STA $DB20,X
1360        INX
1370        CPX #200
1380        BNE BONK
1390 ;
1400 ;
1410 ; INIZIALIZZA LE INTERRUZIONI
1420 ;
1430 LOOP1  SEI
1440        LDA IVEC
1450        STA TEMP
1460        LDA IVEC+1
1470        STA TEMP+1
1480        LDA #<MINE
1490        STA IVEC
1500        LDA #>MINE
1510        STA IVEC+1
1520 ;
1530        LDA #0           ; FERMA IL TIMER A
1540        STA $DC0E
1550        STA FLAG
1560 ;
1570        LDA #49
1580        STA POINT
1590        STA RASTRO
1600 ;
1610        LDA #201
1620        STA POINT+1
1630 ;
1640 ;
1650        LDA #1
1660        STA $D01A        ; ABILITA IL RASTER INTERRUPT
1670        STA $D019        ; COMPARAZIONE DEL RASTER
1680        LDA $D011        ; AZZERA IL BIT PIU´ SIGNIFICATIVO
1690        AND #127
1700        STA $D011
1710 ;
```

```
1720        CLI
1730 ;
1740 ;
1750 ;
1760 ;
1770 CHECK  LDA FLAG2
1780        BPL CHECK
1790        LDA #0
1800        STA FLAG2
1810 ;
1820        LDX #39
1830        LDY #39
1840 SHIFT  LDA (TXTPTR),Y
1850        STA $0770,X
1860        DEX
1870        DEY
1880        BPL SHIFT
1890        INC TXTPTR                                          ALTO
1900        BNE MVTIME      ; SE IL BYTE BASSO DIVENTA 0, INCREMENTA IL BYTE
1910        INC TXTPTR+1
1920 MVTIME LDA TXTPTR      ; SIAMO GIA´ ALLA FINE DEL TESTO?
1930        CMP #<ENDTXT
1940        BNE CHECK
1950        LDA TXTPTR+1
1960        CMP #>ENDTXT
1970        BNE CHECK
1980        LDA #<FRANK     ; RIMETTI IL PUNTATORE ALL´INIZIO
1990        STA TXTPTR
2000        LDA #>FRANK
2010        STA TXTPTR+1
2020        JMP CHECK
2030 ;
2040 FRANK  .BYT ´
2050        .BYT ´QUESTO E´,39,´ UN ESEMPIO DI SCROLL
2060        .BYT ´IN DIREZIONE ORIZZONTALE. UNA VOLTA
2070        .BYT ´CHE TUTTI I DATI SONO STATI MOSTRATI,
2080        .BYT ´LO SCROLL RICOMINCIA ´
2090        .BYT ´DALL´,39,´INIZIO.´
2100 ENDTXT .BYT ´
2110 ;
2120 ; ROUTINE DI SERVIZIO DELLE INTERRUZIONI
2130 ;
2140 MINE   LDA $D019
2150        STA $D019
2160 ;
2170        LDA FLAG        ; FLAG=0 .A=0
2180        EOR #1          ; FLAG=0 .A=1
2190        STA FLAG        ; FLAG=1 .A=1
2200        TAX             ; .X=1 .A=1
2210        LDA POINT,X     ; .X=1 .A=201
2220        STA RASTRO      ; .X=1 .A=201
2230        CPX #1
2240        BNE BOTTOM
2250 ;
2260        LDA $D011       ; MODO BIT MAP
2270        ORA #32
2280        STA $D011       ; BIT MAP ABILITATA
2290        LDA $D018
2300        ORA #$78        ; CARATTERI=$2000+$4000
2310        STA $D018       ; SCHERMO=$1C00+$4000
2320        LDA $D016
2330        ORA #15
2340        STA $D016       ; ATTIVA IL MULTICOLORE
2350 ;
2360        LDA $DD00       ; SELEZIONA IL BANCO 1
2370        AND #254
2380        STA $DD00
2390 ;
2400        LDA SREG        ; IMPOSTA IL REGISTRO DI SCROLL
2410        AND #248
2420        ORA #3
2430        STA SREG
2440 ;
```

```
2450        LDA #14
2460        STA $D021
2470 ;
2480        PLA
2490        TAY
2500        PLA
2510        TAX
2520        PLA
2530        RTI
2540 ;
2550 ;
2560 BOTTOM LDA $D011        ; MODO TESTO
2570        AND #255-32
2580        STA $D011
2590 ;
2600        LDA $DD00        ; CAMBIA NEL BANCO 0
2610        ORA #1
2620        STA $DD00
2630 ;
2640        LDA $D016        ; DISATTIVA IL MULTICOLORE
2650        AND #255-16
2660        STA $D016
2670 ;
2680        LDA #23          ; MEMORIA DI SCHERMO
2690        STA $D018
2700 ;
2710        LDA #0
2720        STA $D021
2730 ;
2740 ;
2750        DEC SCROLL
2760        LDA SCROLL
2770        STA FLAG2
2780        CMP #255
2790        BNE SLURP
2800        LDA #7
2810 SLURP  STA SCROLL
2820        LDA SREG
2830        AND #255-7
2840        ORA SCROLL
2850        STA SREG
2860 BASIRQ JMP (TEMP)
2870 ;
2880        .END

READY.
```

Per leggibilità presentiamo il listato sorgente, con commenti ed etichette, e che potrà essere introdotto con un assemblatore, seguito dal disassemblato, utile per chi abbia a disposizione solo il Monitor di linguaggio macchina.

Ricordate che nel listato sorgente i numeri all'inizio di ogni riga sono proprio numeri di linea, e non indirizzi. Infatti l'indirizzo iniziale del programma è stato specificato nella linea 1120:

*** = $0C00**

Qui di seguito c'è una spiegazione linea per linea del programma, riferendoci al listato sorgente.

La linea 1010 assegna a IVEC il valore $0314, cioè il vettore della richiesta d'interruzione (IRQ). È grazie a questo vettore che il C128 può eseguire il

programma. Infatti, inserendo la vostra routine tra quelle di gestione potete eseguire facilmente e velocemente delle operazioni che normalmente, senza interruzioni, comporterebbero una gran perdita di tempo del processore. Lo schermo viene scandito ogni 1/60 di secondo, ed entro questo tempo avvengono 2 interruzioni: è così possibile avere 2 modi video sullo stesso schermo. Non potete dividere lo schermo senza usare le interruzioni, poichè il processore non è così veloce da seguire il raggio di scansione del video. Per avere un'interruzione ad una determinata riga di schermo, dovete scrivere il numero della riga nel registro di comparazione video in $D012. La linea 1030 assegna questo indirizzo alla variabile RASTRO.

Vedrete più avanti nel programma che il valore posto nel Registro di Comparazione del Raster fa iniziare lo schermo di testo alla riga 201. La prossima interruzione avverrà poi alla riga 49 in cima allo schermo per passare in modo bit map. Tutto ciò si ripete 60 volte al secondo, così l'occhio umano vede due differenti schermi allo stesso tempo.

Continuiamo con la spiegazione. La linea 1040 definisce la variabile BACOL, relativa al registro di fondo #0 in 53281. Nella 1050 viene messo POINT=$1802: POINT e POINT+1 contengono i due valori del raster. In 1060 FLAG è uguagliato a $FC, e servirà al programma per sapere la riga dell'interruzione.

Le linee 1070 e 1080 assegnano a FLAG2 e SCROLL rispettivamente i valori $FD e $FE. FLAG2 contiene lo stesso valore di SCROLL. Ad SREG si assegna $D016, cioè il registro di scroll. Solo i bit da 0 a 2 sono usati per lo scroll, mentre i rimanenti eseguono altri compiti. Sono necessari 3 bit poichè i caratteri son spostati di un pixel alla volta per 7 volte, poi avviene uno spostamento di un intero carattere, e si ricomincia.

La linea 1100 assegna a TXTPTR il valore $FA. TXTPTR punta al testo da visualizzare.

Come detto prima, la linea 1120 specifica l'indirizzo d'inizio del programma. La SYS o il GO del monitor vanno fatti a questo indirizzo.

La prima sequenza di istruzioni pone l'indirizzo del testo (chiamato FRANK) nelle locazioni $FA-FB, chiamate TXTPTR. FRANK è l'etichetta della linea 2040, e segna l'inizio del testo da far scorrere. In questo programma il primo carattere è uno spazio; in effetti i primi 40 caratteri sono degli spazi. Il 41esimo carattere è l'inizio della scritta QUESTO È UN ESEMPIO DI SCROLL, alla linea 2050. I dati dalla linea 2040 alla 2100 sono memorizzati a partire da $0C98, una volta assemblato il sorgente. In questo caso, il byte basso in $FA è $98, mentre il byte alto in $FB è $0C. L'indirizzo completo a 16 bit $0C98 è importante perchè rappresenta l'inizio del testo da far scorrere. Quando il puntatore al testo (TXTPTR) raggiunge la fine del testo, l'indirizzo iniziale $0C98 viene rimesso in TXTPTR.

La seconda sequenza di istruzioni, dalla linea 1190, pone a 7 il valore massimo delle due variabili SCROLL ($FE) e FLAG2 ($FD). Queste saranno usate e spiegate più avanti.

La terza sequenza da 1230 mette in uscita il registro di direzione dati.

Il modulo di istruzioni seguente, da 1280 a 1300, mette lo schermo nel formato a 38 colonne, allo scopo di eseguire lo scroll fine. Il bit 3 di $D016 controlla le colonne: se vale 1 abbiamo 40 colonne, altrimenti 38.

Le colonne ora rimaste ai bordi dello schermo costituiscono lo spazio per i caratteri da far apparire e scomparire un pixel alla volta. Questo programma fa scorrere a sinistra, così i caratteri che devono apparire sono messi sulla colonna 39, prima di passare sulla 38 e diventare visibili. Contemporaneamente, i caratteri sulla colonna 2 passano sulla 1, scomparendo. Ciò succede nelle linee da 1770 a 2020 e sarà spiegato successivamente.

Le linee da 1330 a 1380 riempiono di bianco il quarto inferiore della RAM colore. I 4 bit bassi specificano il colore del carattere.

La memoria di schermo si trova nel banco 0 quando appare il testo in movimento. La pagina grafica e la matrice video sono invece nel banco 1, i 16K da $4000 a $7FFF.

La ragione per cui si riempiono solo le ultime 20 locazioni di RAM colore è perchè solo le ultime 5 righe di testo sono visibili.

Le linee da 1410 a 1700 sono la routine di inizializzazione delle interruzioni. La 1430, etichettata LOOP1, disabilita il riconoscimento delle interruzioni nel registro di stato del processore. Quando questo bit vale 1, il processore non risponderà alle interruzioni. La linea 1720 pone a 0 il bit I e riabilita il riconoscimento delle interruzioni.

Le linee da 1440 a 1470 salvano il contenuto originario del vettore IVEC nelle locazioni TEMP, a cui salterà poi la nostra routine.

Le linee da 1480 a 1510 pongono l'indirizzo iniziale della nostra routine nel vettore IVEC. In questo caso, MINE è l'etichetta all'inizio della routine d'interruzione, in 2140. Nel codice oggetto, il byte basso sarà $70 e quello alto $0D, per formare l'indirizzo $0D70. Una volta messo a 0 il bit I, l'8502 finisce l'esecuzione dell'istruzione corrente e controlla se c'è un segnale d'interrupt. Nel caso, mette a 1 il bit I, in modo da evitare altre interruzioni, poi salva sullo stack il registro di stato P ed il contatore di programma PC. Infine preleva l'indirizzo contenuto in IVEC e salta alla nostra routine.

Le linee 1530 e 1540 fermano il timer A del CIA #1, alla locazione $DC0E. Inoltre, la linea 1550 azzera FLAG. FLAG sarà usata più avanti nel programma per sapere quando è avvenuta l'interruzione.

Le istruzioni da 1570 a 1590 definiscono POINT ($1802) come contenente 49 ($31), il punto dove inizia la bit map. In più inseriscono 49 anche in $D012, il Registro di Comparazione Raster.

Lo schermo a 40 colonne del C128 consiste di 200 righe di raster, ognuna alta un pixel ed avente 320 colonne di pixel. Sapete che il BASIC indirizza le coordinate della bit map su un piano che va da 0,0 nell'angolo superiore sinistro a 319,199 nell'inferiore destro. Invece il raster non ha questi valori. Le righe visibili partono da 50 e vanno fino a 249. Sono gli stessi valori che usano gli sprite. Notate che ci sono sempre 200 righe, ma che rispetto alla bit map partono da 50. Le linee raster inferiori a 50 e maggiori di 249 sono al di fuori del riquadro.

Le istruzioni in 1610 e 1620 pongono 201 ($C9) in POINT+1 ($1803), il valore del raster da dove inizia il modo testo.

Le linee da 1650 a 1670 abilitano l'interruzione raster del video. Ponendo a 1 il bit 0 di questi registri permettiamo al chip VIC di generare l'interruzione da raster una volta raggiunta la riga 49 o la 201.

Le istruzioni delle linee da 1680 a 1700 azzerano il bit più significativo del registro di comparazione raster, in 53265. Questo è un nono bit per arrivare fino a 512 linee di schermo.

L'istruzione della linea 1720 azzera il bit I, permettendo di accettare le interruzioni. Questa è l'ultima istruzione che deve essere eseguita dalla routine di inizializzazione. Ora si è pronti a rispondere alle interruzioni.

Le linee da 1770 a 1800 controllano il valore di FLAG2. Se FLAG2 è positivo, il programma salta all'etichetta della linea 1770 e ricontrolla il valore di FLAG2. Il valore contenuto rappresenta quello dei 3 bit bassi del registro di scroll in $D016. Pure la variabile SCROLL è associata a FLAG2. Nella routine di servizio dell'interruzione (dalla linea 2750 alla 2770), SCROLL viene decrementata e ricopiata in FLAG2. Questo valore è in relazione a quello contenuto in $D016. Ecco la ragione del conteggio:

La direzione dello scorrimento è da destra a sinistra; quindi, dovete porre il valore massimo (7) nei 3 bit bassi di $D016 e decrementarlo. Se al contrario il programma avesse dovuto scrollare da sinistra a destra, avreste dovuto mettere uno 0 ed incrementare. Quando decrementate il registro di scroll, i caratteri nella memoria di schermo sono spostati di 1 pixel a sinistra della posizione precedente, e così per ogni volta successiva. Quando i 3 bit bassi diventano 0, bisogna muovere i caratteri più in giù di una locazione di memoria, così da essere pronti per un nuovo scorrimento fine. La routine va dalla linea 1840 alla 1880. Dopo lo scorrimento di un carattere, i 3 bit bassi sono riportati al valore 7, il VIC sposta di nuovo i caratteri di 7 pixel a sinistra e la vostra routine di un'altra posizione carattere in memoria. Ulteriori dettagli saranno dati nella spiegazione delle linee da 2750 a 2860.

Le istruzioni dalla linea 1820 alla 1880 spostano in su di una locazione di memoria il testo da far scorrere. Prima si caricano i registri X e Y col valore 39 ($27). L'istruzione nella linea 1840 carica il contenuto della locazione puntata da TXTPTR usando l'indirizzamento indiretto indicizzato. L'indirizzo viene calcolato prendendo il contenuto di TXTPTR ($98) ed aggiungendogli Y per arrivare a $BF ($98+$27). L'indirizzo effettivo sarà dunque $0CBF e il primo carattere prelevato uno spazio. Si accederà poi agli elementi successivi decrementando il registro Y.

L'istruzione di memorizzazione della linea 1850 pone il primo carattere del testo nella locazione di schermo 1943 ($0770+$27), che è la 40esima colonna della 23esima riga. Questa colonna non è visibile perchè la larghezza dello schermo è di sole 38 colonne. Ogni nuovo carattere da scrollare deve essere messo in quella posizione per apparire lentamente nella colonna adiacente. Le linee 1860 e 1870 decrementano i registri X e Y, e la 1880 salta all'etichetta SHIFT finchè Y è positivo (uguale o maggiore di 0).

Alla seconda esecuzione del ciclo, l'indirizzo di TXTPTR sarà $0CBE ($0C98+$26), e l'indirizzo dove porre il carattere (sempre uno spazio) sarà 1942 ($0770+$26). La terza volta passerà a 1941 ($0795) e così via. Questo processo continua finchè i registri X ed Y diventano 255, cioè minori di 0. A questo punto è stata posta sullo schermo tutta la serie dei 40 spazi nella linea 2040.

Una volta che X ed Y sono diventati negativi, alla linea 1890 viene incrementato TXTPTR, in modo da puntare un carattere più avanti nel messaggio. Nei primi 40 cicli (dalla linea 1840 alla 1880) sono spostati 40 spazi. Alla prossima esecuzione si sposteranno invece 39 spazi e la lettera Q di QUESTO. Poi 38 spazi e le lettere QU, e così via. Questo processo continua finchè saranno scorse tutte le lettere del messaggio (linee 2040-2100).

La linea 1900 salta all'etichetta MVTIME se TXTPTR (il byte basso dell'indirizzo dei dati) è diverso da 0, altrimenti si incrementa TXTPTR+1 (il byte alto). Le linee da 1920 a 1970 controllano se TXTPTR è arrivato alla fine del messaggio (ENDTXT). Se il puntatore è ancora in mezzo al testo, si salta all'etichetta CHECK, altrimenti, da 1980 a 2010, si rimette il puntatore all'inizio del testo e si ricomincia tutto il processo.

Le linee da 2040 a 2100 sono i dati da far scorrere. Questi dati sono posti in direttive .BYTE, proprie del Sistema di Sviluppo Assembler del Commodore 64. Nel vostro caso, il monitor di linguaggio macchina tratta i dati semplicemente memorizzandoli in locazioni assolute, di cui dovete conoscere a priori l'indirizzo. Nel codice oggetto prodotto dall'assembler i dati vanno da $0C98 a $0D6F, e nel monitor dovrete riferirvi a questi indirizzi e non all'etichetta associata.

LA ROUTINE DI SERVIZIO DELLE INTERRUZIONI

Le istruzioni dalla linea 2140 fino alla fine formano la routine di servizio delle interruzioni. A seconda dei valori del registro di raster vengono eseguite parti differenti del programma. Per esempio, se il raster vale 49 vengono eseguite le linee da 2260 a 2530. Queste selezionano il modo bit map ed eseguono altre funzioni che spiegheremo tra poco. Se il raster vale invece 201, si eseguono le linee da 2560 a 2860. Queste istruzioni, fra l'altro, selezionano il modo carattere standard. Tenete a mente che entrambi i segmenti della routine di servizio vengono eseguiti in una singola scansione dello schermo, ogni 1/60 di secondo. Le istruzioni da 2140 a 2240 sono sempre eseguite ad ogni interruzione.

L'istruzione alla linea 2140 azzera il bit IRQ del registro di comparazione per permettere una nuova interruzione. L'indirizzo dell'etichetta MINE, che viene posto in IVEC nelle linee da 1480 a 1510, dice all'8502 dove inizia la routine di servizio dell'interruzione. Nel codice oggetto questo è l'indirizzo $0D70.

Le linee da 2170 a 2240 determinano la riga del raster a cui è avvenuta l'interruzione. In 2170 si carica in A il valore di FLAG, inizialmente 0. In 2180 si esegue l'OR esclusivo, che tramuta lo 0 in 1. Il valore viene poi rimesso in FLAG. Ad ogni nuova esecuzione, il valore di FLAG passerà da 1 a 0 e viceversa. Alla linea 2200 si ricopia poi A in X. In 2210 si carica in A il valore di POINT o di POINT+1, a seconda del valore di X. Se X vale 0, si carica 49 in A, per specificare la prossima interruzione al raster 49. Se invece X vale 1, viene caricato 201, per porre la prossima interruzione al raster 201. La linea 2220 pone l'accumulatore nell'indirizzo identificato da RASTRO. La linea 2230 confronta X con 1: se X=0 l'interruzione è avvenuta al raster 201 e si salta alla linea 2560; viceversa se X = 1l'interruzione era al raster 49 e il salto non viene eseguito, proseguendo con le linee da 2260 a 2530.

Le istruzioni da 2260 a 2530 eseguono le operazioni relative al modo bit map. Da 2260 a 2280 si seleziona la bit map. Da 2290 a 2310 si pone la matrice video a $1C00 e la bit map a $2000. A entrambi questi indirizzi viene aggiunto lo spostamento obbligatorio di $4000, poichè questo schermo appare nel banco 1. Da 2320 a 2340 si imposta il modo multicolore e da 2360 a 2380 il banco 1. Infine da 2400 a 2430 si impostano al valore 3 i 3 bit bassi del registro di scroll (per avere lo schermo bit map nella posizione normale).

Le istruzioni da 2480 a 2530 ripristinano i valori dei registri A, X ed Y e ritornano dalla routine di servizio.

Le linee da 2560 a 2860 eseguono tutte le operazioni associate al modo testo. Da 2560 a 2580 si sceglie il modo carattere standard. Da 2600 a 2620 si ritorna nel banco video 0, da $0000 a $3FFF. Da 2640 a 2660 si disabilita il multicolore. In 2680-2690 col valore decimale 23 si rimette la locazione dello schermo a 1024 e la memoria caratteri a 6144. Tutti i numeri non preceduti dal segno $ sono assunti come valori decimali in questo particolare assembler. In 2710-2720 si annerisce il fondo dello schermo di testo.

Le linee da 2750 a 2850 regolano il valore del registro di scroll finchè non diventa 0; in tal caso lo si rimette a 7 e FLAG2 diventa 255, per segnalare alla routine nelle linee 1840-1880 di effettuare uno scroll di una posizione carattere.

Infine, la linea 2860 salta al vettore di default dell'IRQ, salvato precedentemente nella variabile TEMP. Ciò permette all'8502 di eseguire la normale routine d'interruzione come se la nostra routine non fosse mai stata eseguita.

Sebbene questo esempio di programma sia lungo e complesso, contiene utili routine e spiegazioni mai apparse in nessun testo della Commodore. Studiate attentamente queste routine e aggiungetele ai vostri programmi. Questa sezione comprende delle informazioni di grande peso per lo sviluppatore di programmi, sia principiante che esperto. Le Figure da 9-28 a 9-32 nelle pagine seguenti forniscono un sommario della programmazione grafica.

	CAMBIO DEI BANCHI VIDEO	SPOSTAMENTO DELLA RAM VIDEO
BASIC C128	**POKE56576,(PEEK(56576)AND 252)ORX** DOVE X È IL VALORE DECIMALE DEI BIT 0 E 1 DELLA TABELLA 9-30	**TESTO** **POKE2604,(PEEK(2604)AND15) ORX** **BIT MAP** **POKE2605,(PEEK(2605)AND15) ORX** DOVE X È UN VALORE DELLA TABELLA 9-29
LINGUAGGIO MACCHINA C128	**LDA $DD00** **AND #$FC** **ORA #$X** **STA $DD00** DOVE X È IL VALORE ESA DEI BIT 0 E 1 DELLA TABELLA 9-30	**TESTO** **LDA $0A2C** **AND #$0F** **ORA #$X** **STA $0A2C** **BIT MAP** **LDA $0A2D** **AND #$0F** **ORA #$X** **STA $0A2D** DOVE X È L'EQUIVALENTE ESA DEL VALORE DECIMALE NELLA TABELLA 9-29
BASIC C64	**POKE56576,(PEEK(56576)AND 252)ORX** DOVE X È IL VALORE DECIMALE DEI BIT 0 E 1 NELLA TABELLA 9-30	**TESTO O BIT MAP** **POKE53272,(PEEK(53272)AND 15)ORX** DOVE X È UN VALORE DECIMALE DELLA FIGURA 9-29
LINGUAGGIO MACCHINA C64	**LDA $DD00** **AND #$FC** **ORA #$X** **STA $DD00** DOVE X È IL VALORE ESA DEI BIT 0 E 1 DELLA TABELLA 9-30	**TESTO O BIT MAP** **LDA $D018** **AND #$0F** **ORA #$X** **STA $D018** DOVE X È L'EQUIVALENTE ESA DEI VALORI DECIMALI IN FIGURA 9-29 IN MODO 64 POTETE DISPORRE DI DUE DIFFERENTI SCHERMI, UNO PER IL TESTO E UNO PER LA BIT MAP, COME FA IL KERNAL DEL C128

Figura 9-28. Sommario della programmazione grafica — PARTE 1

<table>
<tr><th>SPOSTAMENTO
DELLA MEMORIA CARATTERI</th><th>ACCESSO ALLA ROM CARATTERI</th></tr>
<tr><td>TESTO

POKE2604,(PEEK(2604)AND240) ORZ

BIT MAP (SOLO IL BIT 3 HA VALORE)
POKE2605,(PEEK(2605)AND240) ORZ
DOVE Z È UN VALORE DECIMALE
DELLA FIGURA 9-31</td><td>10 BANK14:REM ACCESSO ALLA ROM
20 FORI=0TO7
30 ?PEEK(I);
40 NEXT
50 BANK15</td></tr>
<tr><td>TESTO — BIT MAP (SOLO IL BIT 3 HA VALORE)

LDA $0A2C — LDA $0A2D
AND #$F0 — AND #$F0
ORA #$Z — ORA #$Z
STA $0A2C — STA $0A2D

DOVE Z È L'EQUIVALENTE ESA DEI
VALORI DECIMALI DELLA TABELLA 9-31</td><td>LDA #$01
STA $FF00
LDX #$00
LDA $D000,X
LOOP STA TEMP,X
INX
CPX #$08
BNE LOOP
LDA #$00
STA $FF00</td></tr>
<tr><td>TESTO O BIT MAP 1 (IN BIT MAP SOLO IL BIT 3 HA VALORE)

POKE53272,(PEEK(53272)AND240)
ORZ
DOVE Z È UN VALORE DECIMALE
DELLA FIGURA 9-31</td><td>5 TEMP=6144
10 POKE56334,PEEK(56334)AND254
20 POKE1,PEEK(1)AND251
30 FORI=0TO7
40 POKETEMP+I,PEEK(53248+I)
50 NEXT
60 POKE1,PEEK(1)OR4
70 POKE56334,PEEK(56334)OR1
80 FORI=TEMPTOTEMP+7
90 ?PEEK(I);
100 NEXT</td></tr>
<tr><td>TESTO O BIT MAP (IN BIT MAP SOLO IL BIT 3 HA VALORE)

LDA $D018
AND #$F0
ORA #$Z
STA $D018

DOVE Z È L'EQUIVALENTE ESA DEI
VALORI DECIMALI DELLA FIGURA 9-31</td><td>LDA $DC0E
AND #$FE
STA $DC0E
LDA $01
AND #$FB
STA $01
LDX #$00
LOOP LDA $D000,X
STA TEMP,X
INX
CPX #$08
BNE LOOP
LDA $01
ORA #$04
STA $01
LDA $DC0E
ORA #$01
STA $DC0E</td></tr>
</table>

Figura 9-28. Sommario della programmazione grafica — PARTE 2

LOCAZIONI (RICORDATE LO SPOSTAMENTO DI $4000)

X	BIT	DECIMALE	ESADECIMALE
0	0000----	0	$0000
16	0001----	1024	$0400 (DEFAULT)
32	0010----	2048	$0800
48	0011----	3072	$0C00
64	0100----	4096	$1000
80	0101----	5120	$1400
96	0110----	6144	$1800
112	0111----	7168	$1C00
128	1000----	8192	$2000
144	1001----	9216	$2400
160	1010----	10240	$2800
176	1011----	11264	$2C00
192	1100----	12288	$3000
208	1101----	13312	$3400
224	1110----	14336	$3800
240	1111----	15360	$3C00

Figura 9-29. Locazioni della memoria di schermo

BANCO	INDIRIZZI	VALORE DEI BIT 0 E 1 IN $DD00		
		BINARIO		DECIMALE
0	$0000-$3FFF	11	=	3 (DEFAULT)
1	$4000-$7FFF	10	=	2
2	$8000-$BFFF	01	=	1
3	$C000-$FFFF	00	=	0

Figura 9-30. Indirizzi dei banchi video

LOCAZIONI DELLA MEMORIA CARATTERE
(RICORDATE GLI SPOSTAMENTI DI $4000)

VALORE DI Z	BIT	DECIMALE	ESADECIMALE	
0	----000-	0	$0000-$07FF	
2	----001-	2048	$0800-$0FFF	
4	----010-	4096	$1000-$17FF	IMMAGINI DELLA ROM NEI BANCHI 0 E 2
6	----011-	6144	$1800-$1FFF	(SOLO IN MODO 64)
8	----100-	8192	$2000-$27FF	
10	----101-	10240	$2800-$2FFF	
12	----110-	12288	$3000-$37FF	
14	----111-	14336	$3800-$3FFF	

Figura 9-31. Locazioni della memoria carattere

	DATI DI SCHERMO		DATI COLORE		DATI CARATTERI	
	TESTO	**BIT MAP**	**TESTO**	**BIT MAP**	**TESTO**	**BIT MAP**
BASIC C128 (DEFAULT)	1024-2023 **$0400-07E7** QUESTE LOCAZIONI SONO RILOCABILI	7168-8167 **$1C00-1FE7**	55296-56295 **$D800-DBE7**	MATRICE VIDEO	IMMAGINE ROM **4096-8192** **$1000-1FFF**	8192-16191 **$2000-3F3F** **QUESTE LOCAZIONI SONO RILOCABILI**
LINGUAGGIO MACCHINA C128	**1024-2023** **$0400-07E7** QUESTE LOCAZIONI SONO RILOCABILI	**7168-8167** **$1C00-1FE7**	**55296-56295** **$D800-DBE7**	**MATRICE VIDEO**	**IMMAGINE ROM** **4096-8192** **$1000-1FFF**	**8192 – 16191** **$2000-3F3F** QUESTE LOCAZIONI SONO RILOCABILI
BASIC C64 (DEFAULT)	**1024-2023** **$0400-07E7** QUESTE LOCAZIONI SONO RILOCABILI	**1024-2023** **$0400-07E7**	**55296-56295** **$D800-DBE7**	**MATRICE VIDEO**	**IMMAGINE ROM** **4096-8192** **$1000-1FFF**	NON È DI DEFAULT E DEVE ESSERE LOCATA
LINGUAGGIO MACCHINA C64	**1024-2023** **$0400-07E7** QUESTE LOCAZIONI SONO RILOCABILI	**1024-2023** **$0400-07E7**	**55296-56295** **$D800-DBE7**	**MATRICE VIDEO**	**IMMAGINE ROM** **4096-8192** **$1000-1FFF**	NON È DI DEFAULT E DEVE ESSERE LOCATA

Figura 9-32. Locazioni grafiche di default

NOTA: Queste locazioni fanno riferimento al banco video 0.

10

GLI SPRITE

GLI SPRITE: BLOCCHI DI OGGETTI MOBILI

Uno sprite è un oggetto grafico mobile che potete definire con una forma particolare per la visualizzazione sullo schermo. L'immagine dello sprite può raggiungere la larghezza di 24 pixel per l'altezza di 21 pixel. Ogni pixel corrisponde ad un bit nel campo di memoria dello sprite. Perciò ogni sprite richiede 63 byte di memoria. Il C128 ha locazioni di memoria predefinite per i dati dello sprite nell'intervallo da 3584 ($0E00) fino a 4095 ($0FFF).

Il sistema grafico C128 ha 8 sprite. Ogni sprite si muove sul proprio piano indipendente. Uno sprite si può muovere davanti o dietro agli oggetti o ad altri sprite sullo schermo, dipendendo dalla priorità specificata.Gli sprite grafici standard possono essere di uno qualsiasi dei 16 colori disponibili. Gli sprite multicolori possono avere 3 colori. I colori che vengono assegnati ai pixel nello sprite dipendono dalle formazioni grafiche dell'immagine. Nella memoria dello sprite, i bit inseriti (1) abilitano i pixel dello sprite a visualizzare il colore selezionato dal registro del colore dello sprite; i bit disabilitati (0) disabilitano i pixel corrispondenti dello sprite, rendendoli trasparenti e permettendo così al colore sottostante di passare e di essere visualizzato. Gli sprite possono anche essere espansi fino al doppio della loro grandezza normale sia in direzione verticale che in quella orizzontale.

La maggior parte dei pacchetti di software grafici in commercio per il C128 e C64, si basano sugli sprite. Per le applicazioni grafiche, gli sprite offrono capacità di animazione superiori. Sprite singoli sono utili per piccoli oggetti che si muovono. Potete connettere e sovrapporre parecchi sprite per dare maggiori dettagli ad immagini grafiche animate. Per esempio, supponete di scrivere un programma che animi una persona che corre sullo schermo. Potete fare l'immagine della persona come un singolo sprite, ma l'effetto appare molto più realistico se assegnate sprite diversi per parti diverse del corpo della persona. Le braccia possono essere uno sprite, il corpo un altro, e le gambe un terzo. Poi potete definire due ulteriori sprite: uno come un secondo insieme di gambe in una diversa posizione, e l'altro come un secondo insieme di braccia in una diversa posizione. Posizionate il primo insieme di braccia, il corpo ed il primo insieme di gambe sullo schermo in modo che siano uniti per formare un corpo intero. Continuando ad inserire ed a disinserire i due diversi insiemi di braccia e di gambe, l'immagine appare come se corresse. Questo procedimento comporta il sovrapporre ed il congiungere gli sprite. La spiegazione qui data è un algoritmo semplificato, e la vera programmazione può essere complessa. La programmazione degli sprite è stata resa facile con i nuovi programmi in BASIC 7.0 per gli sprite.

La prima parte di questa sezione spiega i nuovi comandi in BASIC per lo sprite ed illustra la procedura per sovrapporre e congiungere gli sprite, la seconda spiega le operazioni interne degli sprite, l'informazione di memoria, le assegnazioni del colore, l'espansione dello sprite e l'indirizzamento dei registri dello sprite in linguaggio macchina.

BASIC 7.0 I COMANDI DEGLI SPRITE E I LORO FORMATI

Ecco una breve descrizione di ogni comando degli sprite in BASIC 7.0

COLLISION: definisce il tipo di collisione degli sprite sullo schermo, sia sprite contro sprite, che sprite contro un dato di collisione
MOVSPR: posiziona o muove gli sprite da una locazione dello schermo ad un'altra
SPRCOLOR: definisce i colori per gli sprite multicolori
SPRDEF: inserisce il modo definizione dello sprite per editare gli sprite
SPRITE: attiva, colora, imposta le priorità degli sprite sullo schermo ed espande uno sprite
SPRSAV: salva una variabile stringa di un testo in un'area di memoria dello sprite e viceversa o copia i dati da uno sprite ad un altro
SSHAPE: salva una parte dello schermo grafico in una variabile stringa.

COLLISION

Definisce le priorità di collisione dello sprite

COLLISION tipo [,riga di programma]
dove:
tipo tipo di collisione, come segue:
1=collisione di sprite contro sprite
2=collisione di sprite contro un dato dello schermo
3=penna ottica (solo 40 colonne)
riga di programma numero di riga di una subroutine in BASIC

ESEMPIO:

COLLISIONE 1,5000	riconosce una collisione di uno sprite contro uno sprite e programma il controllo mandato alla subroutine alla riga 5000
COLLISIONE 1	ferma un'azione interrotta che era stata inizializzata nell'esempio sopra riportato
COLLISIONE 2,1000	riconosce una collisione tra sprite e dato, e programma il controllo diretto alla subroutine alla riga 1000.

MOVSPR

Posiziona o muove uno sprite sullo schermo (usando uno qualsiasi dei seguenti 4 formati)

1. **MOVSPR numero,X,Y**	pone lo sprite specificato alle coordinate assolute X,Y.
2. **MOVSPR numero,+X,+Y**	muove lo sprite relativo alla sua posizione corrente
3. **MOVSPR numero,X;Y**	muove la distanza X dello sprite all'angolo Y relativo alla sua posizione corrente
4. **MOVSPR numero, angolo#velocità**	muove lo sprite verso un angolo relativo alle sue coordinate originali,in senso orario a velocità specificata.

dove:
numero è il numero dello sprite (da 1 a 8)
X,Y > è la coordinata della locazione dello sprite
ANGOLO è l'angolo (0–360) di movimento in senso orario relativo alla coordinata originale dello sprite
VELOCITÀ è la velocità (0–15) alla quale si muove lo sprite

Posiziona uno sprite in una locazione specifica dello schermo in relazione al piano delle coordinate dello SPRITE, (non al piano dello schermo). MOVSPR inizia anche il movimento dello sprite ad una velocità specificata. Questo capitolo contiene un diagramma del piano di coordinate degli sprite.

ESEMPI:

MOVSPR 1,150,150	posiziona lo sprite 1 alle coordinate 150,150
MOVSPR 1,+20, +30	muove lo sprite 1 a destra di 20 (X) coordinate e giù di 30 (Y) coordinate
MOVSPR 4,50;100	muove lo sprite 4 di 50 coordinate verso un angolo di 100 gradi
MOVSPR 5,45#15	muove lo sprite 5 verso un angolo di 45 gradi in senso orario, relativo alle sue coordinate originali X ed Y. Lo sprite si muove alla velocità più alta (15).

NOTA: Una volta che avete specificato un angolo ed una velocità nella quarta forma della frase MOVSPR, lo sprite continua sulla sua strada (anche se è disabilitato) dopo che il programma si ferma, finchè non ponete la velocità a zero (0) o premete RUN/STOP e RESTORE.

SPRCOLOR

Pone il multicolore 1 e/o il multicolore 2 per tutti gli sprite.

SPRCOLOR [smcr–1] [,smcr–2]

dove:

smcr−1 pone il multicolore 1 per tutti gli sprite
smcr−2 pone il multicolore 2 per tutti gli sprite

Ognuno di questi parametri può essere un qualsiasi colore da 1 a 16.

ESEMPI:

SPRCOLOR 3,7 pone il multicolore 1 dello sprite in rosso ed il multicolore 2 in blu
SPRCOLOR 1,2 pone il multicolore 1 in nero ed il multicolore 2 in bianco

SPRDEF

Imposta il modo DEFinizione dello SPRite per creare immagini dello sprite.

SPRDEF

Il comando SPRDEF definisce gli sprite interattivamente.
Impostando il comando SPRDEF, si visualizza un'area di lavoro dello sprite sullo schermo larga 24 caratteri ed alta 21 caratteri. Ogni posizione di un carattere nella griglia corrisponde ad un pixel dello sprite nello sprite visualizzato a destra dell'area di lavoro. Segue un riassunto delle operazioni del modo DEFinizione dello SPRite e dei tasti che le eseguono. L'area di DEFinizione dello SPRite è mostrata nella Figura 10−1.

INPUT DELL'UTENTE	DESCRIZIONE
tasti 1−8	**selezionano un numero di sprite solo alla richiesta NUMERO DI SPRITE ? (SPRITE NUMBER ?)**
A	**movimento automatico di accensione e di spegnimento del cursore**
tasti CRSR	**muove il cursore**
RETURN	**muove il cursore all'inizio della riga successiva**
RETURN	**esce dal modo disegno dello sprite solo alla richiesta NUMERO DI SPRITE ?**
HOME	**sposta il cursore nell'angolo a sinistra in alto dell'area di lavoro dello sprite**
tasto CLR	**cancella l'intera griglia**
tasti 1−4	**selezionano le sorgenti di colore ed abilitano i pixel dello sprite**
tasto CTRL,1−8	**seleziona il colore di fondo dello sprite (1−8)**
tasto Commodore,1−8	**seleziona il colore di fondo dello sprite (9−16)**
tasto STOP	**cancella i cambiamenti e ritorna alla richiesta**
SHIFT RETURN	**salva lo sprite e ritorna a NUMERO DI SPRITE ?**
X	**espande lo sprite nella direzione X (orizzontale)**
Y	**espande lo sprite nella direzione Y (verticale)**
M	**sprite multicolore**
C	**copia i dati dello sprite da uno sprite all'altro**

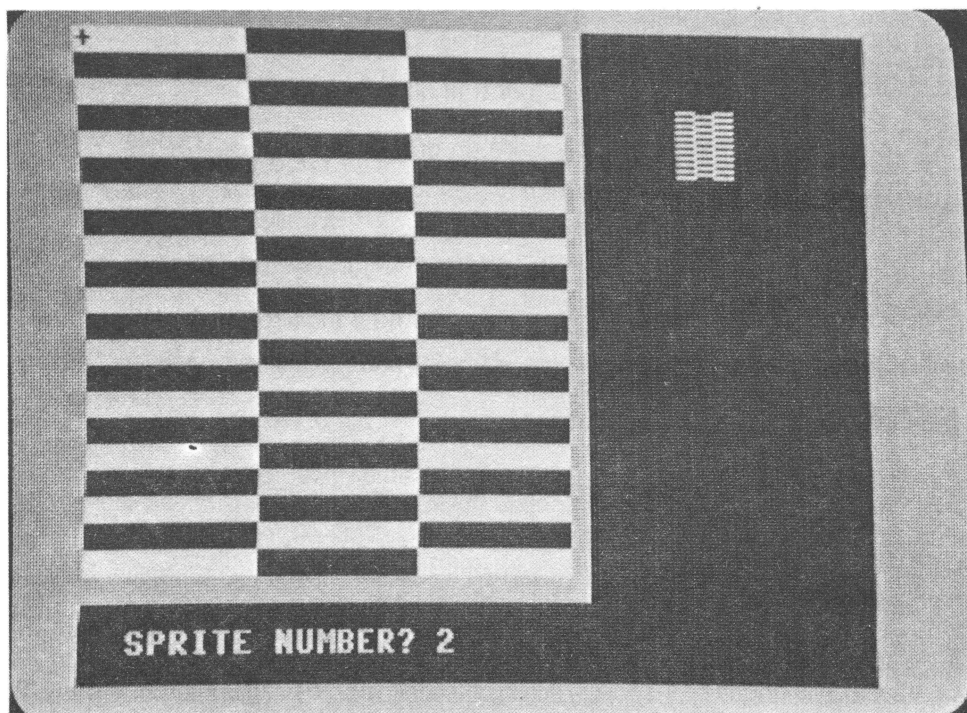



Figura 10-1. Area di DEFinizione dello SPRite

PROCEDURA DI CREAZIONE DELLO SPRITE IN MODO DEFINIZIONE DELLO SPRITE

Ecco la procedura generale per creare uno sprite in modo definizione dello sprite:

1. Azzerate l'area di lavoro premendo i tasti shift e CLR—HOME contemporaneamente
2. Se volete uno sprite multicolore, premete il tasto M ed il cursore (+) appàre grande il doppio dell'originale.
   Il cursore appare grande il doppio poichè il modo multicolore accende due pixel per ognuno nel modo standard dello sprite. Gli sprite multicolori hanno solo metà della risoluzione orizzontale degli sprite standard.
3. Selezionate un colore dello sprite. Per i colori tra 1 e 8 tenete abbassato il tasto CONTROL e premete un tasto tra 1 e 8. Per selezionare codici di colori tra 9 e 16, tenete abbassato il tasto COMMODORE e premete un tasto tra 1 e 8.

4. Ora siete pronti per creare la forma del vostro sprite. I tasti numerati da 1 a 4 riempiono lo sprite e gli danno la forma. Per uno sprite monocolore usate il tasto 2 per riempire un carattere nell'area di lavoro. Premete il tasto 1 per cancellare quello che avete disegnato con il tasto 2. Se volete riempire un carattere alla volta, premete il tasto A. Ora dovete muovere il cursore manualmente con i tasti del cursore. Se volete che il cursore si muova automaticamente verso destra mentre lo tenete abbassato, premete ancora il tasto A. Se riempite un carattere nell'area di lavoro, potete vedere il pixel corrispondente accendersi nello sprite visualizzato.
   L'immagine dello sprite cambia non appena editate l'area di lavoro. Nel modo multicolore il tasto 2 riempie due caratteri nell'area di lavoro con il multicolore 1, il tasto 3 riempie due caratteri con il multicolore 2. Potete spegnere (colorare il pixel con il colore di fondo) le aree riempite nell'area di lavoro con il tasto 1.
   Nel modo multicolore il tasto 1 spegne due caratteri alla volta.
5. Mentre state costruendo il vostro sprite, vi potete muovere liberamente nell'area di lavoro senza accendere o spegnere nessun pixel, usando i tasti RETURN, HOME ed il cursore.
6. In qualsiasi momento potete espandere il vostro sprite sia in direzione verticale che in quella orizzontale.
   Per espanderlo verticalmente, premete il tasto Y. Per espanderlo orizzontalmente, premete il tasto X. Per tornare alla grandezza normale dello sprite visualizzato, premete ancora il tasto X o Y. Quando un tasto accende e spegne lo stesso controllo, si dice che è bistabile, così i tasti X ed Y bistabilizzano l'espansione verticale ed orizzontale dello sprite.
7. Quando avete terminato di creare il vostro sprite e siete soddisfatti del suo aspetto, salvatelo in memoria tenendo premuto il tasto SHIFT e premendo il tasto RETURN. Il Commodore 128 memorizza i dati dello sprite nell'appropriata area di memoria dello sprite. Lo sprite visualizzato nell'angolo in alto a destra dello schermo è spento ed il comando è tornato alla richiesta NUMERO DI SPRITE. Se volete creare un altro sprite inserite un altro numero dello sprite ed editate il nuovo sprite proprio come avete fatto con il primo.
   Se volete visualizzare ancora lo sprite originale nell'area di lavoro, inserite il numero dello sprite originale. Per uscire dal modo DEFinizione dello SPRite, basta premere RETURN alla richiesta NUMERO DI SPRITE.
8. Potete copiare uno sprite dentro un altro con il tasto COMMODORE.
9. Se non volete SALVARE il vostro sprite, premete il tasto STOP. Il Commodore 128 spegne lo sprite visualizzato ed ogni modifica che voi avete fatto viene cancellata. Siete ritornati alla richiesta NUMERO DI SPRITE.
10. Per uscire dal modo DEFinizione dello SPRite, premete il tasto RETURN mentre la richiesta NUMERO DI SPRITE è visualizzata sullo schermo, senza che sia seguita da un numero di sprite. Potete uscire con entrambe le seguenti condizioni:

- Immediatamente dopo avere SALVATO il vostro sprite in memoria (shift RETURN)
- Immediatamente dopo avere premuto il tasto STOP

Una volta che avete creato uno sprite e siete usciti dal modo DEFinizione dello SPRite, il vostro dato dello sprite è memorizzato nell'appropriata area di memoria dello sprite nella memoria del Commodore 128. Poichè ora siete tornati in linguaggio BASIC, dovete accendere il vostro sprite per vederlo sullo schermo.

Per accenderlo usate il comando SPRITE che avete imparato.

Per esempio, avete creato lo sprite 1 in modo SPRDEF. Per accenderlo in BASIC, coloratelo in blu ed espandetelo sia in direzione X che in direzione Y ed inserite questo comando:

SPRITE 1,1,7,0,1,1,0

Ora usate il comando MOVSPR per muoverlo di un angolo di 90 gradi alla velocità di 5, nel seguente modo:

MOVSPR 1,90#5

Ora sapete tutto del modo SPRDEF. Prima create lo sprite, salvate i dati dello sprite ed uscite dal modo SPRDEF passando al BASIC. Poi accendete il vostro sprite con il comando SPRITE. Muovetelo con il comando MOVSPR. Quando avete finito la programmazione, SALVATE i dati del vostro sprite in un file binario con il comando BSAVE come segue:

BSAVE "nome del file",B0,P3584 TO P4096 (che salva tutti gli 8 sprite).

SPRITE

Accende, spegne, colora, espande e regola le priorità di uno sprite sullo schermo.

SPRITE numero> [,acceso/spento] [,fngd] [,priorità] [,x-esp] [,y-esp] [,modo]

La frase SPRITE controlla la maggior parte delle caratteristiche di uno sprite. Le parentesi significano dei parametri opzionali. Se omettete un parametro dovete includere ancora una virgola al suo posto.

PARAMETRO	DESCRIZIONE
numero	**numero dello sprite (1 – 8)**
acceso – spento	**accendere sprite (1) o spegnere (0)**
fondo	**colore di fondo dello sprite (1 – 16)**
priorità	**la priorità è 0 se gli sprite appaiono davanti agli oggetti sullo schermo. La priorità è 1 se gli sprite appaiono dietro agli oggetti sullo schermo.**
x-esp	**espansione orizzontale accesa (1) o spenta (0)**
y-esp	**espansione verticale accesa (1) o spenta (0)**
modo	**seleziona uno sprite standard (0) o uno sprite multicolore (1)**

I parametri non specificati in frasi successive degli sprite assumono le caratteristiche della frase precedente SPRITE. Potete verificare le caratteristiche di uno sprite con la funzione RSPRITE.

ESEMPI:

SPRITE 1,1,3	accende lo sprite numero 1 e lo colora di rosso
SPRITE 2,1,7,1,1,1	accende lo sprite numero 2, lo colora di blu, lo fa passare dietro agli oggetti sullo schermo e lo espande nelle direzioni verticale ed orizzontale
SPRITE 6,1,1,0,0,1,1	accende SPRITE numero 6, lo colora di nero.Il primo 0 dice al computer di visualizzare gli sprite davanti agli oggetti sullo schermo. Il secondo 0 e l'1 seguente dicono al C128 di espandere lo sprite solo verticalmente. L'ultimo 1 specifica il modo multicolore. Usate il comando SPRCOLOR per selezionare i multicolori dello sprite.

SPRSAV

Memorizza i dati dello sprite da una variabille stringa del testo in un'area di memoria dello sprite o viceversa.

SPRSAV origine>, destinazione>

Questo comando copia un'immagine dello sprite da una variabile stringa ad un'area di memoria dello sprite. Copia anche i dati dall'area di memoria dello sprite in una variabile stringa. Sia l'origine che la destinazione possono essere un numero di sprite o una variabile stringa, ma non possono essere entrambe variabili stringa. Per copiare una stringa in uno sprite, si usano solo i primi 63 byte dei dati. Il resto è ignorato poichè uno sprite può mantenere solo 63 data byte.

ESEMPI:

SPRSAV 1,A$	copia la forma grafica dallo sprite 1 alla variabile stringa A$
SPRSAV B$,2	copia i dati dalla variabile stringa B$ allo sprite 2
SPRSAV 2,3	copia i dati dallo sprite 2 allo sprite 3.

SSHAPE

Salva/carica le forme a/da variabili stringa.

SSHAPE e GSHAPE vengono utilizzati per salvare e caricare aree rettangolari di schermi multicolori o bit-map a/da variabili stringhe in BASIC. Il comando per salvare un'area dello schermo in una variabile stringa è:

SSHAPE variabile stringa,X1,Y1 [,X2,Y2]

dove:

variabile stringa	nome della stringa dove salvare i dati
X1,Y1	coordinate dell'angolo (0,0 fino a 319,199) (in scala)
X2,Y2	coordinate dell'angolo opposte (X1,Y1) (il CP, cursore pixel, lo assume per default)

Vedete anche il comando LOCATE descritto nel Volume 2 Capitolo 4 per informazioni sul cursore pixel.

ESEMPI:

SSHAPE A$,10,10	salva un'area rettangolare dalle coordinate 10,10 alla locazione del CP, nella variabile stringa A$
SSHAPE B$,20,30,47,51	salva un'area rettangolare dalle coordinate in alto a sinistra (20,30) attraverso le coordinate in basso a destra (47,51) nella variabile stringa B$
SSHAPE D$,+10,+10	salva un'area rettangolare di 10 pixel verso destra e di 10 pixel in basso dalla posizione corrente del cursore pixel.

CONNESSIONI DI SPRITE

Il programma seguente è un esempio di connessione di sprite. Il programma crea un ambiente spaziale esterno. Disegna stelle, un pianeta ed un veicolo spaziale simile ad Apollo. La navicella viene disegnata e poi memorizzata in due stringhe di dati, A$ e B$. La parte anteriore della navicella, la capsula, è memorizzata nello sprite 1. La metà retrostante della navicella, il retrorazzo, è memorizzata nello sprite 2. La navicella vola lentamente attraverso lo schermo due volte. Poichè viaggia così lentamente ed è molto lontana dalla terra, è necessario che venga lanciata in direzione della terra con i retrorazzi. Dopo il secondo viaggio attraverso lo schermo, i retrorazzi si accendono e spingono la capsula in salvo verso la terra. Ecco il listato del programma:

```
5 COLOR 4,1:COLOR 0,1:COLOR 1,2:REM SELEZIONE DEL COLORE NERO PER LO SFONDO,
  DEL BIANCO PER IL PRIMO PIANO
10 GRAPHIC 1,1:REM SCELTA DELL'ALTA RISOLUZIONE
17 FOR I=1 TO 40
18 X=INT(RND(1)*320)+1:REM DISEGNA LE STELLE
19 Y=INT(RND(1)*200)+1:REM DISEGNA LE STELLE
21 DRAW 1,X,Y:NEXT    :REM DISEGNA LE STELLE
22 BOX 0,0,5,70,40,,1:REM PULISCE IL RIQUADRO
```

```
23 BOX 1,1,5,70,40
24 COLOR 1,8:CIRCLE 1,190,90,35,25:PAINT 1,190,95:REM TRACCIA E COLORA IL PIANETA
25 CIRCLE 1,190,90,65,10:CIRCLE 1,190,93,65,10:CIRCLE 1,190,95,65,10:COLOR 0,1
26 DRAW 1,10,17 TO 32,10 TO 33,20 TO 32,30 TO 16,23 TO 10,23 TO 10,17
28 DRAW 1,19,24 TO 20,21 TO 27,25 TO 26,28:REM FINESTRA FINALE
35 DRAW 1,20,19 TO 20,17 TO 29,13 TO 30,18 TO 28,23 TO 20,19:REM FINESTRA INIZIALE
38 PAINT 1,13,20:REM COLORA L'ASTRONAVE
40 DRAW 1,34,10 TO 36,20 TO 34,30 TO 45,30 TO 46,20 TO 45,10 TO 34,10:REM SP1
42 DRAW 1,45,10 TO 51,12 TO 57,10 TO 57,17 TO 51,15 TO 46,17:REM ENG1
43 DRAW 1,46,22 TO 51,24 TO 57,22 TO 57,29 TO 51,27 TO 45,29:REM ENG2
44 PAINT 1,40,15:PAINT 1,47,12:PAINT 1,47,26:DRAW 0,45,30 TO 46,20 TO 45,10
45 DRAW 0,34,14 TO 44,14 :DRAW 0,34,21 TO 44,21:DRAW 0,34,28 TO 44,28
47 SSHAPE A$,10,10,33,32:REM SALVATAGGIO SPRITE IN A$
48 SSHAPE B$,34,10,57,32:REM SALVATAGGIO SPRITE IN B$
50 SPRSAV A$,1:REM DATI SPR1
55 SPRSAV B$,2:REM DATI SPR2
60 SPRITE 1,1,3,0,0,0,0:REM DEFINISCE I VALORI PER LO SPR1
65 SPRITE 2,1,7,0,0,0,0:REM DEFINISCE I VALORI PER LO SPR2
82 MOVSPR 1,150 ,150:REM POSIZIONE INIZIALE DI SPR1
83 MOVSPR 2,172 ,150:REM POSIZIONE INIZIALE DI SPR2
85 MOVSPR 1,270 # 5 :REM MUOVE SPR1 LUNGO LO SCHERMO
87 MOVSPR 2,270 # 5 :REM MUOVE SPR2 LUNGO LO SCHERMO
90 FOR I=1 TO 5950:NEXT:REM RITARDO
92 MOVSPR 1,150,150:REM POSIZIONE DI SPR1 PER IL LANCIO DEL RETRORAZZO
93 MOVSPR 2,174,150:REM POSIZIONE DI SPR2 PER97 FOR I=1 TO 1200:NEXT:REM RITARDO
98 SPRITE 2,0:REM TURNO DEL RETRORAZZO (SPR2)
99 FOR I=1 TO 20500:NEXT:REM RITARDO
100 GRAPHIC 0,1:REM RITORNO ALLA MODALITA' TESTO
```

Ecco la spiegazione del programma:

Riga 5 COLORA il fondo di nero ed il primo piano di bianco
Riga 10 seleziona il modo bit—map standard e schiarisce lo schermo grafico
Righe 17-21 disegnano le stelle
Riga 23 incasella in un'area del video la figura della navicella nell'angolo in alto a sinistra dello schermo
Riga 24 disegna e colora i pianeti
Riga 25 disegna i cerchi intorno al pianeta
Riga 26 disegna il contorno della capsula della navicella
Riga 28 disegna la finestra inferiore della capsula spaziale
Riga 35 disegna la finestra superiore della capsula spaziale
Riga 38 colora la capsula di bianco
Riga 40 disegna il contorno del retrorazzo della navicella
Righe 42 e 43 disegnano i motori del retrorazzo sul retro della navicella
Riga 44 colora i motori del retrorazzo e disegna il contorno posteriore del retro del retrorazzo con il colore di fondo
Riga 45 disegna delle righe nella parte del retrorazzo della navicella col colore di fondo. (A questo punto voi visualizzate solo figure sullo schermo. Non avete usato alcuna frase sprite, così la vostra navicella non è ancora uno sprite).
Riga 47 posiziona le coordinate SSHAPE sulla prima metà (24 per 21 pixel) della capsula della navicella e la memorizza in una stringa di dati A$
Riga 48 posiziona le coordinate SSHAPE sulla seconda metà (24 per 21 pixel) della navicella e la memorizza in una stringa di dati B$
Riga 50 trasferisce i dati da A$ allo sprite 1
Riga 55 trasferisce i dati da B$ allo sprite 2
Riga 60 accende lo sprite 1 e lo colora di rosso

Riga 65 accende lo sprite 2 e lo colora di blu
Riga 82 posiziona lo sprite 1 alle coordinate 150,150
Riga 83 posiziona lo sprite 2 a 23 pixel a destra delle coordinate di partenza dello sprite 1
Righe 82 e 83 collegano realmente i due sprite
Righe 85 e 87 muovono gli sprite uniti attraverso lo schermo
Riga 90 ritarda il programma. Questa volta il ritardo è necessario perchè gli sprite completino i due viaggi attraverso lo schermo. Se tralasciate il ritardo, gli sprite non hanno abbastanza tempo per muoversi attraverso lo schermo
Righe 92 e 93 posizionano gli sprite al centro dello schermo, e preparano la navicella ad accendere i retrorazzi
Riga 95 aziona lo sprite 1, la capsula spaziale, in avanti. Il numero 10 alla riga 95 specifica la velocità con la quale lo sprite si muove. La velocità va da 0 (stop) a 15 (velocità massima)
Riga 96 muove la parte spenta del retrorazzo della navicella indietro e fuori dallo schermo
Riga 97 è un altro ritardo così il retrorazzo, lo sprite 2, ha il tempo di uscire dallo schermo
Riga 98 spegne lo sprite 2, una volta fuori dallo schermo
Riga 99 è un altro ritardo così la capsula può continuare a muoversi attraverso lo schermo
Riga 100 vi riporta allo stato testo.

ESEMPI DI PROGRAMMI DI SPRITE

Il modo migliore di creare gli sprite è con lo SPRDEF. I seguenti esempi danno per scontato che voi abbiate creato i vostri sprite in modo SPRDEF. Il primo esempio di programma sprite illustra l'uso dei comandi SPRITE e MOVSPR. Posiziona tutti gli otto sprite così che appaiano convergere in una locazione dello schermo, poi li manda in tutte otto le direzioni. Ecco il listato:

```
10 REM ESEMPIO DI MOVIMENTO DEGLI SPRITE
20 FOR I=1 TO 8
30 MOVSPR I,100,100
40 NEXT
50 FOR I=1 TO 8
60 SPRITE I,1,I,1,1,1,0
70 MOVSPR I,I*30 # I
80 NEXT
```

Dalla riga 20 alla 40 posizionano tutti 8 gli sprite alla locazione sprite di coordinate 100,100. A questo punto gli sprite non sono ancora abilitati, ma quando lo sono, tutti otto sono uno sopra l'altro.

Le righe 50 e 60 accendono ognuno degli 8 sprite in 8 colori diversi. La prima "I" è il parametro del numero di sprite. Il primo "1" alla riga 60 è l'abilitazione di ogni sprite. La seconda "I" specifica il codice colore per ogni sprite. Il secondo "1" (il quarto parametro) pone le priorità di visualizzazione per ogni sprite. Una priorità di visualizzazione 1 dice al C128 di visualizzare gli sprite dietro agli oggetti dello schermo. Una priorità video zero abilita gli sprite a passare davanti agli oggetti del testo o dello schermo bit map. Il sesto ed il settimo parametro, entrambi sono uno (1), espandono la grandezza dello sprite sia in direzione verticale che in orizzontale del doppio della grandezza originale. L'ultimo parametro della frase SPRITE seleziona il modo di visualizzazione grafica per gli sprite; sia sprite bit map standard (0) che sprite bit map multicolori (1). In questo esempio, gli sprite sono visualizzati come sprite bit map standard.

La riga 70 muove gli sprite sullo schermo. Il primo parametro, I, rappresenta il numero di sprite. Il secondo parametro,"I*30", definisce l'angolo col quale lo sprite viaggia sullo schermo. Il segno # significa che gli sprite si muovono secondo un particolare angolo e velocità. L'ultimo parametro "I" specifica la velocità alla quale gli sprite viaggiano sullo schermo. In questo esempio, lo sprite 1 si muove alla velocità minima 1, lo sprite 2 si muove alla velocità successiva 2, mentre lo sprite 8 muove il più veloce degli 8 sprite a velocità 8. La velocità più alta a cui uno sprite si può muovere è 15. Infine, la riga 80 completa la struttura FOR...NEXT del ciclo. Notate che gli sprite si muovono continuamente anche dopo che il programma ha smesso il RUNning. Il motivo è che gli sprite sono inseriti nel processo di interruzione del C128. Per spegnere e fermare gli sprite sullo schermo, emettete un comando SPRITE che li spenga o premete RUN/STOP e RESTORE.

Il secondo esempio di programma di sprite dà un algoritmo semplificato di connessione di sprite. Muove due sprite uniti attraverso lo schermo con un angolo di 90 gradi, considerando che i vostri sprite già risiedano nell'intervallo di memoria dello sprite tra 3584 ($0E00) e 4095 ($0FFF). Per semplicità, se non avete immagini di sprite memorizzate nell'area dati dello sprite, riempite l'area dati dello sprite con 255 ($FF) dall'interno del Monitor di Linguaggio Macchina con questo comando:

```
F 0E00 0FFF FF
```

Questo comando accende tutti i pixel all'interno di ogni sprite. Ora potete vedere come l'algoritmo unificante pone e muove gli sprite 7 e 8 fianco a fianco. Ecco il listato:

```
10 REM SPRITE CONTIGUI IN MOVIMENTO
20 REM IL PROGRAMMA ASSUME I VALORI DEGLI SPRITE RESIDENTI NELLA MEMORIA STESSA
30 I=1 : REM INIZIALIZZA LA DISTANZA I
35 SCNCLR
40 MOVSPR 8,50,100:REM POSIZIONE INIZIALE DELLO SPRITE 8
50 MOVSPR 7,73,100:REM POSIZIONE INIZIALE DELLO SPRITE 7 CONTIGUO ALLO SPRITE 8
60 DO
70 SPRITE 8,1,3:REM ATTIVA LO SPRITE 8
80 SPRITE 7,1,4:REM ATTIVA LO SPRITE 7
90 MOVSPR 8,I;90:REM MUOVE LO SPRITE 8 DI I UNITA' CON ANGOLO DI 90 GRADI
100 MOVSPR 7,I;90:REM MUOVE LO SPRITE 7 DI I UNITA' CON ANGOLO DI 90 GRADI
110 I=I+1: REM INCREMENTO DELLA DISTANZAI
120 LOOP
```

La riga 30 inizializza la variabile distanza I a 1.

La riga 40 posiziona lo sprite 8 alle coordinate assolute 50,100. Poichè questo programma muove due sprite uniti da sinistra a destra con un angolo di 90 gradi, lo sprite 7, che è attaccato allo sprite 8, deve essere posizionato in modo che tocchi il lato destro dello sprite 8. La riga 50 pone lo sprite 7 esattamente sul lato destro dello sprite 8. Poichè uno sprite è largo 24 pixel (prima dell'espansione), per unire due sprite, ponete lo sprite da unire esattamente 24 pixel a destra della posizione delle coordinate dell'angolo in alto a sinistra dello sprite 8. La posizione di uno sprite è posta sul piano delle coordinate dello sprite secondo l'estremo pixel in alto a sinistra dello sprite. Poichè la posizione originale dello sprite 8 è 50,100, sommate 24 (incluso) alla coordinata X (orizzontale) per fare coincidere i lati di entrambi gli sprite. Questo purchè i vostri sprite siano larghi esattamente 24 pixel. Se non riempite interamente le dimensioni di uno sprite, dovrete aggiustare le coordinate per farli incontrare correttamente.

A questo punto, le coordinate degli sprite coincidono perfettamente. La riga 60 comincia un ciclo, così la distanza può essere aggiornata per abilitare il movimento degli sprite attraverso lo schermo. Le righe 70 ed 80 abilitano gli sprite 8 e 7 e li colorano rispettivamente di rosso e azzurro.

Le righe 90 e 100 muovono gli sprite 8 e 7 rispettivamente con un angolo di 90 gradi secondo la distanza specificata dalla variabile I. La riga 110 aggiorna la distanza di 1 ad ogni giro per la durata del ciclo. La riga 120 fa girare il ciclo finchè la variabile distanza I è uguale a 320.

Il terzo esempio di sprite dà un algoritmo per sovrapporre due sprite e muoverli sullo schermo con un angolo di 45 gradi. Questo programma considera un'altra volta che i dati dei vostri sprite siano nella memoria dello sprite. Se non lo fossero, riempite gli sprite con i dati come nell'ultimo esempio di connessione. Ecco il listato:

```
10 REM ESEMPIO DI SOVRAPPOSIZIONE
20 REM QUESTO PROGRAMMA ASSUME I VALORI DEGLI SPRITE RESIDENTI NELLA MEMORIA
   STESSA DEGLI SPRITE
30 I=1 :REM INIZIALIZZA LA DISTANZA I
35 SCNCLR
40 MOVSPR 8,50,100:REM SETTA LA POSIZIONE INIZIALE DELLO SPRITE 8
50 MOVSPR 7,50,100:REM SETTA LA POSIZIONE INIZIALE DELLO SPRITE 7 CON LA
   COPERTURA DELLO SPRITE 8
60 DO
70 SPRITE 8,1,3    :REM ATTIVA SPR 8
80 MOVSPR 8,I;45   :REM MUOVE LO SPR 8 DI I UNITA' CON  ANGOLAZIONE DI 45 GRADI
90 SPRITE 8,0,3    :REM TURNO DELLO SPR 8
100 SPRITE 7,1,4   :REM ATTIVA SPR 7
110 MOVSPR 7,I;45  :REM MUOVE LO SPR 7 DI I UNITA' CON  ANGOLAZIONE DI 45 GRADI
120 SPRITE 7,0,3   :REM TURNO DELLO SPR 7
140 LOOP
```

Come nell'ultimo programma, la riga 30 inizializza la variabile distanza I a 1. Le righe 40 e 50 posizionano gli sprite 8 e 7, rispettivamente, alle coordinate 50, 100. A questo punto gli sprite non sono ancora abilitati, ma quando lo saranno, lo sprite 7 si sovrapporrà allo sprite 8 poichè il numero più basso di sprite ha priorità video sul numero più alto di sprite.

La riga 60 inizia un ciclo DO per muovere gli sprite sul piano di coordinate degli sprite. La riga 70 abilita lo sprite 8 e lo colora di rosso. La riga 80 lo muove di una coordinata secondo il valore corrente di I. La riga 90 disabilita lo sprite 8.

Le righe da 100 a 120 fanno per lo sprite 7 le stesse operazioni delle righe da 70 a 90 per lo sprite 8: abilitano, muovono di una singola coordinata secondo il valore I e disabilitano. La riga 140 ripete il procedimento.

Questo procedimento è ripetuto così velocemente sembra che i due sprite alternino i movimenti. Quando create le immagini reali per il vostro programma di sovrapposizione degli sprite, le immagini che alternerete saranno quelle che simuleranno il movimento di due immagini e creeranno un'immagine animata.

Create due sprite che formino una singola immagine animata. Potreste dover perfezionare la temporizzazione di abilitazione e disabilitazione delle immagini per fare apparire più omogenea l'immagine animata, ma avete una base per l'animazione di due oggetti in un singolo oggetto in movimento.

Questi esempi di programma sono scritti in BASIC, ma gli algoritmi sono gli stessi sia che voi stiate programmando in BASIC o in linguaggio macchina. La prossima parte tratta le operazioni sprite indipendentemente dal linguaggio BASIC. Poichè questa parte ha spiegato gli sprite secondo il BASIC, la prossima continuerà ad elaborare le operazioni interne agli sprite da una prospettiva a livello (linguaggio) macchina.

LE OPERAZIONI INTERNE DEGLI SPRITE

Avete visto come creare, muovere, colorare ed espandere gli sprite con i comandi sprite del BASIC 7.0. Questa parte spiega come controllare gli sprite fuori dai comandi sprite in BASIC (tranne lo SPRDEF). Cioè quali registri VIC siano simulati e gli specifici bit che devono essere attivati o cancellati per manipolare le funzioni sprite.

I registri del chip VIC controllano tutti gli aspetti degli sprite. L'abilitazione di bit specifici in certi registri VIC attiva le funzioni degli otto sprite disponibili. L'ordine con cui attivate queste funzioni è critico per l'animazione dello sprite. Segue un elenco dei passi necessari per visualizzare, colorare, muovere ed espandere gli sprite. Accanto ad ogni passo c'è il registro del chip VIC o la locazione di memoria coinvolta in ogni elemento della programmazione degli sprite.

Questi registri controllano la maggior parte delle caratteristiche degli sprite. Una volta imparati questi passi di programmazione, sarete in grado di avere pieno controllo del video e del movimento degli sprite.

SEQUENZA PROGRAMMAZIONE SPRITE	REGISTRI COINVOLTI
1. crea l'immagine dello sprite	Memoria Dati dello Sprite: 3584-4095 ($0E00-$0FFF). Anche questo è programmabile. Dovete cambiare i valori riferimento degli sprite.
2. indica i dati degli sprite	Indicatori Dati degli sprite: 2040-2047 ($07F8-$07FF), o 8184-8191 ($1FF8-$1FFF) quando lo schermo bit-map è stato azzerato con GRAPHIC 1,1.
3. abilita (accendi) lo sprite	53269 ($D015) (7-0 bit, dipende dal numero di sprite)
4. colora lo sprite	Standard 53287-53294 ($D027-$D02E) Multicolore 53276 ($D01C), 53285 ($D025), 53286 ($D026)
5. posiziona lo sprite	53248-53264 ($D000-$D010)
6. espandi lo sprite	53271 ($D017) (direzione Y), 53277 ($D01D) (direzione X)
7. definisci le priorità video dello sprite	53275 ($D01B)
8. definisci le priorità di collisione dello sprite	53278 ($D01E), 53279 ($D01F)

CREAZIONE DELL'IMMAGINE

Un modo facile per creare gli sprite sul C128 è con il modo DEFinizione degli SPRite (SPRDEF). Per una spiegazione dello SPRDEF vedete la sezione SPRDEF all'inizio di questo capitolo. Questa sezione considera che l'immagine del vostro sprite sia già creata, e che risieda nell'area di memoria dello sprite. Prima di lasciare lo SPRDEF, ricordate di premere i tasti SHIFT e RETURN contemporaneamente; così lo SPRDEF memorizza i dati dello sprite nell'area di memoria dello sprite. Premete ancora RETURN per uscire dallo SPRDEF.

Il Commodore 128 ha una porzione di memoria apposita che va dall'indirizzo decimale 3584 ($0E00) a 4095 ($0FFF), dove il dato dello sprite è memorizzato. Questa porzione di memoria occupa 512 byte. Come sapete uno sprite ha la larghezza di 24 pixel per l'altezza di 21 pixel. Negli sprite standard, ogni pixel corrisponde ad un bit in memoria. Se il bit in uno sprite è spento (uguale a 0), il pixel corrispondente sullo schermo è trasparente, cosa che permette allo sfondo di passare attraverso lo sprite. Se un bit nello sprite è acceso (uguale a 1), il pixel corrispondente sullo schermo è acceso col colore di sfondo come determinato dai registri del colore dello sprite.

La combinazione di zeri e uno produce l'immagine che vedete sullo schermo. Gli sprite multicolori assegnano i colori in modo diverso. Per i dettagli vedete la sezione sugli sprite multicolori più avanti in questo capitolo. Poichè uno sprite è 24 per 21 pixel ed ogni pixel è rappresentato da un bit in memoria, uno sprite usa 63 byte di memoria. Osservate la Figura 10-2 per capire i fabbisogni di memoria per un dato dello sprite.

```
          12345678   12345678   12345678
1         ........   ........   ........
2         ........   ........   ........
3         ........   ........   ........
4         ........   ........   ........
5         ........   ........   ........
6         ........   ........   ........
7         ........   ........   ........
8         ........   ........   ........
9         ........   ........   ........
10        ........   ........   ........
11        ........   ........   ........
12        ........   ........   ........
13        ........   ........   ........
14        ........   ........   ........
15        ........   ........   ........
16        ........   ........   ........
17        ........   ........   ........
18        ........   ........   ........
19        ........   ........   ........
20        ........   ........   ........
21        ........   ........   ........
Ogni       = 24 bit   = 3 byte
Riga
```

Figura 10-2. Fabbisogni dei Dati degli Sprite

Uno sprite richiede 63 byte di dati. Ogni blocco di sprite è formato da 64 byte; il byte extra non viene usato. Poichè il Commodore 128 ha otto sprite ed ognuno consiste in un blocco di sprite da 64 byte, il computer ha bisogno di 512 (8 per 64) byte per rappresentare i dati di tutte otto le immagini dello sprite. L'area dove tutti gli otto blocchi dello sprite risiedono comincia alla locazione di memoria 3584 ($0E00) e finisce alla locazione 4095 ($0FFF). La Figura 10-3 lista gli intervalli di indirizzo della memoria dove ogni sprite memorizza i suoi dati.

```
$0FFF  (4095 Decimale)
       }—Sprite 8
$0FC0
       }—Sprite 7
$0F80
       }—Sprite 6
$0F40
       }—Sprite 5
$0F00
       }—Sprite 4
$0EC0
       }—Sprite 3
$0E80
       }—Sprite 2
$0E40
       }—Sprite 1
$0E00  (3584 Decimale)
```

Figura 10-3. Fasce di Indirizzo di Memoria per la Memorizzazione degli Sprite

Tenete presente che ci si riferisce agli sprite da 1 a 8 in BASIC, ma da 0 a 7 in linguaggio macchina.

INDICATORI DI SPRITE

Il chip VIC ha bisogno di sapere dove cercare i bit grafici (dati) che creano l'immagine dello sprite in memoria. Gli indicatori di sprite sono usati proprio a questo scopo. Diversamente dal Commodore 64, il C128 ha gli indicatori di dati degli sprite automaticamente riempiti con i valori che portano il chip VIC ad indicare i dati memorizzati nell'intervallo dello sprite 3584 ($0E00) fino a 4095 ($0FFF). Questi indicatori di dati sprite sono localizzati da 2040 ($07F8) a 2047 ($07FF) per gli sprite 0 e 7 rispettivamente. Essi sono localizzati anche nell'intervallo di indirizzo da 8184 ($1FF8) a 8191 ($1FFF), una volta che lo schermo grafico è azzerato con il comando GRAPHIC 1,1. I contenuti default di queste locazioni sono:

Esadecimali	38 39 3A 3B 3C 3D 3E 3F
Decimali	56 57 58 59 60 61 62 63

Per trovare la locazione reale dove gli indicatori dati degli sprite cercano i dati in memoria, moltiplicate i contenuti dei dati sprite per 64 (decimali). Moltiplicando questi valori vedrete che gli indicatori cercano i dati nelle locazioni default di memoria degli sprite nella Figura 10-3. Osservate la Figura 10-4 per una spiegazione.

Il modo in cui il Commodore 128 indica automaticamente il dato corretto dello sprite è conveniente per la programmazione, poichè elimina un passo (se i valori originari degli indicatori di sprite non sono stati modificati). Se volete memorizzare i dati degli sprite in qualche altra parte della memoria, dovete cambiare il valore originale dell'indicatore dello sprite (dalla locazione 2040 alla 2047, o dalla 6184 alla 8191) con un valore uguale a:

Inizio dati dello sprite/64= nuovi contenuti dell'indicatore dello sprite

DATI SPRITE	**CONTENUTI PUNTATORE DEI DATI****	**INIZIO DEGLI**
Puntatore del Dato dello Sprite 0	**=56**	*** 64=3584 ($0E00)**
Puntatore del Dato dello Sprite 1	**=57**	*** 64=3648 ($0E40)**
Puntatore del Dato dello Sprite 2	**=58**	*** 64=3712 ($0E80)**
Puntatore del Dato dello Sprite 3	**=59**	*** 64=3776 ($0EC0)**
Puntatore del Dato dello Sprite 4	**=60**	*** 64=3840 ($0F00)**
Puntatore del Dato dello Sprite 5	**=61**	*** 64=3904 ($0F40)**
Puntatore del Dato dello Sprite 6	**=62**	*** 64=3968 ($0F80)**
Puntatore del Dato dello Sprite 7	**=63**	*** 64=4032 ($0FC0)**

** =Solo per il banco del video 0.

Figura 10-4. Locazioni dei Dati degli Sprite

L'inizio dei dati dello sprite è diviso per 64 perchè l'area dati è localizzata in una sezione di 64 byte. Per esempio, se volete porre il dato 0 dello sprite nella nuova locazione 6144 ($1800), dividete 6144 per 64 ed otterrete 96. Ponete il valore 96 ($60) nell'indirizzo 2040 ($078F).

ABILITAZIONE DI UNO SPRITE

Una volta che l'immagine dello sprite è stata definita, e che l'indicatore dati indica il dato giusto, potete accendere lo sprite. Fate questo ponendo un valore nel Registro di Abilitazione dello Sprite, alla locazione 53269 ($D015). Il valore posto nel registro dipende da quale/quali sprite volete accendere. I bit da 0 a 7 corrispondono agli sprite da 0 a 7. Per abilitare lo sprite 0 ponete il bit 0. Per abilitare lo sprite 1, ponete il bit 1 e così via. Il valore che ponete nel registro abilitatore dello sprite è uguale a due elevato alla posizione del bit in decimali.

Se state programmando in linguaggio macchina e volete abilitare più di uno sprite per volta, sommate i valori di due elevato alle posizioni dei bit corrispondenti e memorizzate il risultato nel registro abilitatore dello sprite. Per esempio, per abilitare lo sprite 5, elevate due alla quinta potenza (32($20)) e memorizzatelo come segue:

```
LDA #$20
STA $D015
```

Per abilitare gli sprite 5 e 7, elevate due alla quinta (32($20)) e sommatelo a due alla settima (128($80)) per ottenere il risultato 160 ($A0):

```
LDA $A0
STA $D015
```

La rappresentazione binaria nel Monitor di Linguaggio Macchina è il modo più semplice per capire quanto detto:

```
LDA # % 10100000
STA $D015
```

Per disabilitare la visualizzazione degli sprite, cancellare i bit nel registro abilitatore dello sprite.

ASSEGNAZIONE DEL COLORE AGLI SPRITE

Gli sprite hanno due tipi di visualizzazione del colore: bit-map standard e multicolore. L'assegnazione del colore ai pixel negli sprite funziona in modo simile ai modi bit-map standard e multicolore per lo schermo.

SPRITE BIT-MAP STANDARD

Ogni sprite bit-map standard ha il proprio registro del colore. I quattro bit più bassi di ogni registro del colore determinano il colore dello sprite come specificato dai sedici codici dei colori C128. La figura 10-5 mostra i registri del colore dello sprite bit-map standard.

INDIRIZZO		DESCRIZIONE
53287	($D027)	REGISTRO COLORE DELLO SPRITE 0
53288	($D028)	REGISTRO COLORE DELLO SPRITE 1
53289	($D029)	REGISTRO COLORE DELLO SPRITE 2
53290	($D02A)	REGISTRO COLORE DELLO SPRITE 3
53291	($D02B)	REGISTRO COLORE DELLO SPRITE 4
53292	($D02C)	REGISTRO COLORE DELLO SPRITE 5
53293	($D02D)	REGISTRO COLORE DELLO SPRITE 6
53294	($D02E)	REGISTRO COLORE DELLO SPRITE 7

Figura 10-5. Registri dei Colori degli Sprite Bit Map Standard

La figura 10-6 elenca i codici dei colori posti nei registri del colore dello sprite bit-map standard:

0	Nero	8	Arancione
1	Bianco	9	Marrone
2	Rosso	10	Rosso Chiaro
3	Azzurro	11	Grigio Scuro
4	Porpora	12	Grigio
5	Verde	13	Verde Chiaro
6	Blu	14	Blu Chiaro
7	Giallo	15	Grigio Chiaro

Figura 10-6. Codici dei Colori degli Sprite

Negli sprite bit-map standard, i dati nel blocco dello sprite determinano il modo in cui i colori vengono assegnati ai pixel sullo schermo entro lo sprite visibile. Se il bit nella memoria del blocco è 1, al pixel corrispondente sullo schermo è assegnato il colore dal registro del colore dello sprite standard. Se il bit nel blocco di dati dello sprite è zero, quei pixel sullo sprite sono trasparenti ed i dati di fondo passano dallo schermo allo sprite.

SPRITE MULTICOLORI

Gli sprite multicolori offrono un grado di libertà nell'uso del colore, ma ulteriori colori vanno a scapito di una più alta risoluzione degli sprite standard. Gli sprite multicolori sono visualizzati in tre colori più il colore di fondo. Agli sprite bit-map multicolori sono assegnati colori come agli altri modi multicolori. Prima che possiate assegnare colori multipli ad uno sprite, dovete abilitare lo sprite multicolore. Il registro dello sprite multicolore in locazione 53276 ($D01C) opera

come il registro abilitatore dello sprite. I bit da 0 a 7 corrispondono agli sprite da 0 a 7. Per selezionare uno sprite multicolore, ponete il numero di bit che corrisponde al numero di sprite. Questo richiede che voi eleviate due alla posizione del bit dello sprite che volete visualizzato in multicolore. Per esempio, per selezionare lo sprite 4 a sprite multicolore, elevate due alla quarta potenza (16) e ponetelo nel registro multicolore dello sprite. In linguaggio macchina, seguite le seguenti istruzioni:

```
LDA #$10
STA $D01C
```

Per selezionare più di uno sprite multicolore, sommate i valori di due elevato alle posizioni dei bit e memorizzate il valore nel registro multicolore dello sprite. Il chip VIC fornisce due registri multicolori (0 e 1), nei quali porre i codici dei colori. Queste sono le locazioni dei registri multicolori degli sprite:

	INDIRIZZO
Registro Multicolore 0	**53285($D025)**
Registro Multicolore 1	**53286($D026)**

I codici del colore sono quelli elencati nella Figura 10-6.

Come il modo carattere multicolore, ai pixel negli sprite multicolori vengono assegnati i colori secondo le forme dei bit nel blocco di memoria dello sprite. In questo modo, i bit nel blocco dello sprite sono raggruppati a coppie. La coppia di bit determina come ai pixel siano assegnati i loro colori individuali:

COPPIA DI BIT	**DESCRIZIONE**
00	**TRASPARENTE (COLORE DELLO SCHERMO)**
01	**REGISTRO MULTICOLORE DELLO SPRITE #0 (53285) ($D025)**
10	**REGISTRO COLORE DELLO SPRITE**
11	**REGISTRO MULTICOLORE DELLO SPRITE #1 (53286) ($D026)**

Se la coppia di bit è 00, i pixel sono trasparenti e lo sfondo dello schermo passa attraverso lo sprite. Se la forma dei bit è 10 (binario), il colore viene preso dal registro colore dello sprite (locazione 53287-53294) dello sprite che viene definito. Altrimenti, le altre due possibilità di coppie di bit (01 e 11) vengono prese dal registro multicolore 0 e 1 rispettivamente.

POSIZIONAMENTO DEGLI SPRITE SULLO SCHERMO

Ogni sprite ha due registri posizione per il controllo della posizione dello sprite sullo schermo visibile: le posizioni orizzontale (coordinata X) e verticale (coordinata Y). La figura 10-7 dà le locazioni di memoria del registro posizione dello sprite come appaiono nella memoria del C128.

LOCAZIONE DECIMALE	LOCAZIONE ESADECIMALE	DESCRIZIONE
53248	($D000)	REGISTRO DI POSIZIONE X DELLO SPRITE 0
53249	($D001)	REGISTRO DI POSIZIONE Y DELLO SPRITE 0
53250	($D002)	REGISTRO DI POSIZIONE X DELLO SPRITE 1
53251	($D003)	REGISTRO DI POSIZIONE Y DELLO SPRITE 1
53252	($D004)	REGISTRO DI POSIZIONE X DELLO SPRITE 2
53253	($D005)	REGISTRO DI POSIZIONE Y DELLO SPRITE 2
53254	($D006)	REGISTRO DI POSIZIONE X DELLO SPRITE 3
53255	($D007)	REGISTRO DI POSIZIONE Y DELLO SPRITE 3
53256	($D008)	REGISTRO DI POSIZIONE X DELLO SPRITE 4
53257	($D009)	REGISTRO DI POSIZIONE Y DELLO SPRITE 4
53258	($D00A)	REGISTRO DI POSIZIONE X DELLO SPRITE 5
53259	($D00B)	REGISTRO DI POSIZIONE Y DELLO SPRITE 5
53260	($D00C)	REGISTRO DI POSIZIONE X DELLO SPRITE 6
53261	($D00D)	REGISTRO DI POSIZIONE Y DELLO SPRITE 6
53262	($D00E)	REGISTRO DI POSIZIONE X DELLO SPRITE 7
53263	($D00F)	REGISTRO DI POSIZIONE Y DELLO SPRITE 7
53264	($D010)	REGISTRO MSB DELLO SPRITE X

Figura 10-7. Locazione di Memoria delle Posizioni dei Registri degli Sprite

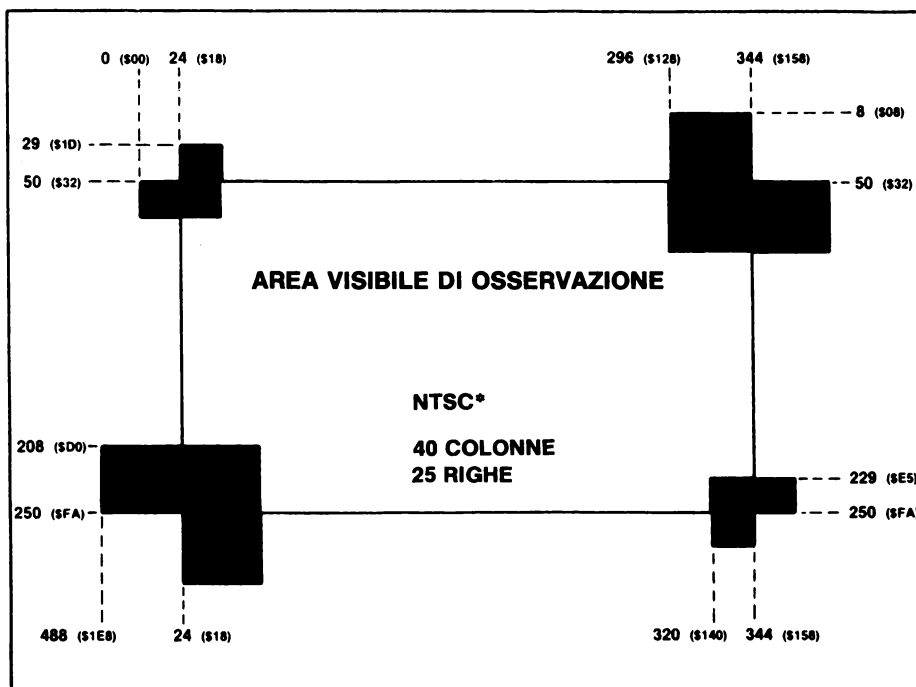
I registri di posizione dello sprite insieme pongono lo sprite su una coordinata orizzontale e su una verticale. La posizione di riferimento per la coordinata verticale e per l'orizzontale è presa dal pixel nell'angolo in alto a sinistra all'interno dello sprite. Tutte le volte che volete porre lo sprite in una particolare posizione dello schermo, calcolate la posizione usando il pixel nell'angolo in alto a sinistra all'interno dello sprite. Il piano di coordinate dello sprite non è lo stesso del piano di coordinate bit-map. Il piano di coordinate bit-map comincia nell'angolo all'estrema sinistra in alto dello schermo alle coordinate 0,0. L'angolo più basso a destra del piano delle coordinate bit-map è il punto 319,199. Il piano di coordinate dello sprite comincia nel punto 24,50 nell'angolo in alto a sinistra dello schermo visibile. L'ultimo punto visibille nel piano coordinate dello sprite nell'angolo in basso a destra dello schermo è 343,249. La Figura 10-8 mostra in che modo il piano delle coordinate dello sprite è in rapporto con lo schermo visibile.

* Standard televisivo di trasmissione Nordamericana per televisione domestica.

Figura 10-8. Coordinate Visibili dello Sprite

Dopo avere visto il piano delle coordinate dello sprite, avrete notato qualcosa di insolito. Le posizioni delle coordinate verticali hanno una fascia di 200, quelle orizzontali hanno una fascia di 320 coordinate. Poichè il C128 è un computer da 8 bit, il valore più alto che qualsiasi registro possa rappresentare è 255.

Come è possibile posizionare uno sprite dopo la 255esima posizione orizzontale dello schermo? Occorre prendere in prestito un bit da un altro registro per rappresentare un valore più grande di 255.

C'è già un bit extra a parte nella memoria del Commodore 128 nel caso in cui vogliate muovere uno sprite dopo la 255esima coordinata orizzontale. La locazione 53264 controlla il movimento dello sprite dopo la posizione 255. Ognuno degli 8 bit in 53264 controlla uno sprite. Il bit 0 controlla lo sprite 0, il bit 1 controlla lo sprite 1, e così via. Per esempio, se si pone il bit 7, lo sprite 7 può muoversi dopo la 255esima posizione orizzontale.

Ogni volta che volete che uno sprite si muova attraverso l'intero schermo, accendete il bit preso in prestito alla locazione 53264 quando lo sprite raggiunge la posizione orizzontale 255. Una volta che lo sprite si allontana dall'angolo destro dello schermo, spegnete il bit preso in prestito, così lo sprite può tornare al lato sinistro dello schermo. I seguenti comandi permettono allo sprite sette di muoversi oltre la 255esima posizione orizzontale:

```
LDA $D010
ORA #$80
STA $D010
```

Il numero 128 è il valore decimale risultante dall'aver posto il bit 7. Ottenete questo valore elevando due alla settima potenza. Se volete abilitare il bit 5, elevate due alla quinta potenza, che, naturalmente, è 32. La regola generale è elevare due alla potenza del numero di sprite che volete muovere dopo la 255esima posizione orizzontale dello schermo. Ora potete prendere in prestito il bit extra di cui avete bisogno per muovere uno sprite tutto intorno allo schermo. Per permettere allo sprite di riapparire sul lato sinistro dello schermo, spegnete ancora il bit sette, come segue:

```
LDA$D010
AND #$7F
STA $D010
```

Non tutte le posizioni orizzontali (X) e verticali (Y) sono visibili sullo schermo. Solo dalla posizione verticale 50 alla 249 e dalla posizione orizzontale 24 alla 343 sono visibili. La locazione 0,0 è fuori dallo schermo come lo è qualsiasi locazione orizzontale minore di 24 e maggiore di 343. Anche qualsiasi locazione verticale minore di 50 e maggiore di 249 è fuori dallo schermo. Le locazioni fuori schermo sono poste da parte così che un'immagine animata può muoversi tranquillamente dentro e fuori lo schermo.

ESPANSIONE DELL'AMPIEZZA DELLO SPRITE

Il chip VIC offre una funzione che permette agli sprite di espandersi in ampiezza in entrambe le direzioni orizzontale e verticale. Quando lo sprite è espanso, la risoluzione dello sprite non aumenta, i pixel all'interno dello sprite coprono proprio due volte tale area; perciò, lo sprite è grande il doppio. Ecco le locazioni in memoria per l'espansione verticale ed orizzontale dello sprite:

	INDIRIZZO
Registro di Espansione Verticale (Y) dello Sprite	**53271 ($D017)**
Registro di Espansione Orizzontale (X) dello Sprite	**53277 ($D01D)**

Questi registri operano nella stessa maniera del registro abilitatore dello sprite. I bit da 0 a 7 corrispondono agli sprite da 0 a 7. Per espandere la misura dello sprite in entrambe le direzioni, elevate due alla posizione del bit e ponetelo nei registri dell'espansione. Per esempio, per espandere lo sprite 7 in entrambe le direzioni, eseguite queste istruzioni in linguaggio macchina:

```
LDA #$80 (%10000000 = notazione binaria nel Monitor)
STA $D017
STA $D01D
```

Per espandere più di uno sprite, sommate due elevato alla posizione del bit per i numeri degli sprite che volete espandere e memorizzate il risultato nei registri dell'espansione. Per fare ritornare gli sprite alla loro misura originaria, azzerate i bit nei registri di espansione.

PRIORITÀ DI VISUALIZZAZIONE DEGLI SPRITE

Nei vostri programmi sprite, avete l'opzione di visualizzare gli sprite davanti o dietro altri sprite o oggetti sullo schermo. Ciò è noto come definizione delle priorità di visualizzazione dello sprite. Il chip VIC definisce due diverse priorità di visualizzazione per gli sprite:

1. Sprite con sprite
2. Sprite con dato

PRIORITÀ DI VISUALIZZAZIONE SPRITE/SPRITE

Ad ognuno degli otto sprite disponibili sul C128 è assegnato il proprio piano nel quale muoversi indipendentemente dagli altri sprite o dalle figure sullo sfondo dello schermo. Visualizzate i piani degli sprite come nella Figura 10-9.

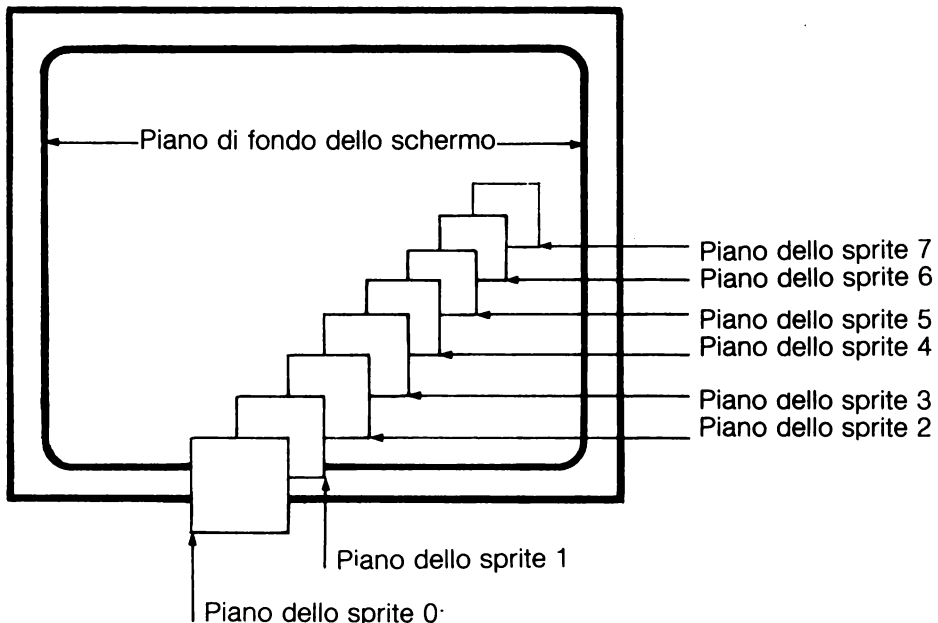



Figura 10-9. Relazioni tra i Piani degli Sprite

La priorità di visualizzazione di ogni piano dello sprite dipende dal numero degli sprite. Le priorità di visualizzazione sprite/sprite sono predefinite dal numero di sprite. Questa funzione è presente nell'hardware del C128, e non è controllata da un registro software. Più basso è il numero di sprite, maggiore è la priorità. Lo sprite con priorità maggiore passa davanti allo sprite con priorità minore. Lo sprite 0 ha la priorità maggiore; perciò, passa davanti a qualsiasi altro sprite sullo schermo se si incontrano in una locazione dello schermo. Per esempio, gli sprite 1 e 5 si muovono verso una stessa locazione dello schermo; quando la raggiungono, lo sprite 1 passa davanti allo sprite 5, poichè lo sprite con numero inferiore ha priorità maggiore di visualizzazione. Tenetelo a mente quando volete che gli sprite si incontrino e passino dietro o davanti l'un l'altro. È importante quando gli sprite si sovrappongono. Assegnate allo sprite che volete passi davanti ad altri sprite il numero più basso di sprite.

Le porzioni di sprite (pixel) a cui non è assegnato un colore (i bit nel blocco di memoria dello sprite corrispondenti a quei pixel sono uguali a zero) sono trasparenti. I buchi in questi sprite permettono al dato dello sfondo di passare attraverso l'area trasparente e di creare un effetto "finestra". Lo stesso effetto finestra capita quando uno sprite a priorità maggiore con "buchi" passa davanti ad uno sprite con priorità inferiore. Anche se uno sprite ha priorità maggiore, se sue porzioni sono trasparenti, lo sprite a priorità inferiore o i dati di visualizzazione dello sfondo passano attraverso lo sprite a priorità maggiore.

PRIORITÀ DI VISUALIZZAZIONE SPRITE/DATO

La priorità sprite/dato (schermo di fondo) è selezionata con un registro software nel chip VIC. Il Registro di Priorità di Visualizzazione Sprite/Sfondo (locazione 53275 ($D01B)) specifica se uno sprite passa davanti o dietro agli oggetti sul piano di sfondo dello schermo. Questa funzione fa sembrare gli sprite più realistici. Il valore di default di questo registro è zero; perciò tutti gli sprite passano davanti agli oggetti sullo sfondo dello schermo a meno che cambiate i valori dei bit in questo registro. In altre parole, all'inserimento, tutti gli sprite hanno una priorità maggiore dello sfondo.

I bit da 0 a 7 corrispondono agli sprite da 0 a 7. Se si pone un bit in questo registro, lo sprite passa dietro gli oggetti sullo sfondo dello schermo. La corrispondenza del numero di bit con il numero degli sprite funziona allo stesso modo del registro abilitatore dello sprite ed a molti degli altri registri dello sprite. Per porre questi bit, elevate due alla posizione del bit per ogni sprite che volete far passare dietro agli oggetti sullo sfondo dello schermo. Per esempio, per fare passare lo sprite 6 dietro gli oggetti sullo sfondo dello schermo, elevate due alla sesta potenza e ponete l'esadecimale equivalente nella locazione 53275 ($D01B) in linguaggio macchina come segue:

LDA #$40 (%01000000= rotazione binaria nel Monitor)
STA $D01B

Per porre le priorità di visualizzazione sprite/dato per più di uno sprite, sommate i valori di due elevato alle rispettive posizioni dei bit e memorizzate il risultato alla locazione 53275 ($D01B).

PRIORITÀ DI COLLISIONE DI SPRITE

Il chip VIC ha una funzione che vi abilita a riconoscere quando avviene una collisione tra sprite, o tra uno sprite e gli oggetti dello schermo.

COLLISIONI DI SPRITE CONTRO SPRITE

Una collisione sprite contro sprite avviene quando un pixel abilitato sullo sfondo di uno sprite si sovrappone ad un pixel abilitato sullo sfondo di un altro sprite in un qualsiasi punto del piano delle coordinate dello sprite. La collisione può anche avvenire in una locazione fuori dallo schermo. La locazione 53278 ($D01E) segnala se è avvenuta una collisione sprite contro sprite. Questo registro, come la maggior parte dei registri sprite, ha un bit che riconosce una collisione per ogni sprite. I bit da 0 a 7 corrispondono agli sprite da 0 a 7. Se uno sprite è coinvolto in una collisione sprite contro sprite, il bit corrispondente allo sprite coinvolto nella collisione viene caricato; perciò, in una collisione sprite contro sprite sono sempre caricati almeno due bit. Questi bit rimangono caricati finchè non vengono letti, momento in cui il VIC cancella il registro. Dovreste memorizzare il valore di questo registro in una variabile finchè la collisione o il codice condizionato che dipende dalla collisione è completamente finita.

Una volta che una collisione sprite contro sprite viene riconosciuta, viene posto il segnale di Richiesta di Interruzione della collisione sprite contro sprite (IRQ), bit 2 di locazione 53273 ($D019), ed avviene un'interruzione se abilitata nel Registro Maschera IRQ a 53274 ($D01A). Quando succede questo, potete incorporare una routine interruzione che possa essere attivata alla collisione di due sprite. Perciò la vostra routine interruzione della collisione sprite contro sprite viene eseguita solamente alla condizione che due sprite collidano. Questa funzione incorporata vi dà modo di inserire, in modo da condizionarlo, una routine interruzione (IRQ) nel vostro programma applicativo, a seconda del comportamento degli sprite sullo schermo.

COLLISIONI DI SPRITE CONTRO DATO

Una collisione di sprite contro dato avviene quando un pixel abilitato sullo sfondo dello sprite si sovrappone ad un pixel dello sfondo di un oggetto sullo schermo. La locazione 53279 ($D01F) dà un segnale quando avviene una collisione sprite contro dato. Questo registro ha un bit che riconosce una collisione per ogni sprite. I bit da 0 a 7 corrispondono agli sprite da 0 a 7. Se uno sprite è coinvolto in una collisione sprite contro dato, il bit corrispondente allo sprite coinvolto nella collisione è attivato. Questi bit rimangono attivati finchè sono letti, a questo punto il chip VIC cancella l'intero registro. Si raccomanda come pratica di programmazione di memorizzare il valore di questo registro in una variabile finchè la collisione o il codice condizionato che dipende dalla collisione sia completamente terminato.

Una volta che una collisione sprite contro dato viene rilevata, il segnalatore della collisione sprite contro dato (IRQ) è attivato nel bit 1 di locazione 53273

($D019) ed avviene un'interruzione se abilitata nel Registro di Maschera IRQ a 53274 ($D01A). Quando questo avviene, potete incorporare una routine interruzione che venga attivata alla collisione di due sprite. Perciò la vostra routine di interruzione della collisione tra sprite e dato viene eseguita solo alla condizione che uno sprite abbia colliso con un oggetto sullo sfondo dello schermo. Ancora, questo vi dà un modo di introdurre, per condizionare, una routine IRQ nel vostro programma di applicazione a seconda del comportamento dei vostri oggetti grafici animati sullo schermo.

Notate che le collisioni di sprite contro dati non avvengono con le coppie di bit multicolori 01 (binarie). Questo permette che quei bit siano interpretati come dati di visualizzazione dello sfondo, senza interferire con le collisioni sprite contro dati.

11

PROGRAMMAZIONE DEL CHIP A 80 COLONNE (8563)

Il computer Commodore 128 offre due tipi di output video: a 40 colonne, video composito con il chip VIC, ed a 80 colonne, RGBI con il chip 8563. Il video ad 80 colonne aggiunge un'importante funzione alla famiglia Commodore di calcolatori domestici e da ufficio: il C128 può essere considerato macchina per l'ufficio. Il chip 8563 abilita il C128 a visualizzare fogli elettronici, wordprocessor, gestioni di database, ed applicazioni CP/M ad 80 colonne. Ora, l'ultimo nella famiglia dei computer economici Commodore esegue la serie Perfect di applicazioni gestionali, e molte altre applicazioni gestionali in modo C128. In modo CP/M, il C128 esegue Wordstar, e molte altre popolari applicazioni gestionali. Inoltre, il C128 sostiene tutto l'hardware ed il software disponibile per il Commodore 64. Il Commodore 128 è realmente il personal computer completo.

LE FUNZIONI VIDEO DEL CHIP 8563

Lo scopo primario del chip video 8563 è di visualizzare i caratteri sullo schermo. L'8563 ha due insiemi di caratteri, ognuno di 256 elementi. Diversamente dal chip VIC, tuttavia, l'8563 può visualizzare tutti i 512 caratteri contemporaneamente. Il chip VIC visualizza solo un insieme di caratteri per volta.

Il chip 8563 sostiene un modo bit-map limitato. La grafica può essere ottenuta con i vostri programmi, preferibilmente in linguaggio macchina. I comandi grafici del BASIC 7.0 non sono adatti al video ad 80 colonne. La programmazione dello schermo grafico in BASIC non è consigliata, poichè il linguaggio non è equipaggiato per manipolare singoli bit video alla volta. Più avanti in questo capitolo, la grafica dello schermo ad 80 colonne è illustrata in linguaggio macchina. Un'altra funzione dell'8563 è lo spostamento del contenuto dello schermo (scrolling) nelle direzioni verticale ed orizzontale. Il chip 8563 ha un insieme di registri scrolling che permette al testo di essere spostato in alto, in basso, a sinistra ed a destra, come vedremo più avanti nel capitolo.

PROGRAMMAZIONE DEL CHIP A 80 COLONNE (8563)

La programmazione del video chip 8563 è molto diversa dalla programmazione del chip VIC. Come sapete, i registri del chip VIC sono localizzati nell'intervallo da 53248 ($D000) a 53296 ($D030) nel banco 15. Diversamente dal chip VIC, l'8563 ha solo due locazioni di memoria nella memoria I/O del Commodore 128; $D600 e $D601. Ciò significa che solo due locazioni di memoria nella memoria I/O del Commodore 128 corrispondono al video chip 8563. L'8563 ha 37 registri interni, sebbene non siano indirizzabili nella memoria I/O del C128. Inoltre, l'8563 ha 16K di RAM che sono indipendenti dalla RAM del Commodore 128. Dovete indirizzarvi alle locazioni $D600 e $D601 come accessi attraverso i quali vi indirizzate indirettamente ai 37 registri interni ed ai 16K di RAM dell'8563. Non potete accedere direttamente a nessuno dei 37 registri interni o ai 16K di RAM dell'8563.

La locazione $D600 è il Registro Indirizzo, ed il $D601 è il Registro Dati. Di solito ponete un numero di registro 8563 nel registro indirizzo ($D600) poi o scrivete o leggete a/dal registro dati nella locazione $D601. Questa è una spiegazione semplificata: per programmare l'8563 con successo, sono date nelle parti seguenti informazioni più dettagliate.

NOTE sull'8563. 1. non potete usare le istruzioni in BASIC PEEK, POKE, o WAIT per accedere all'8563, perchè questi comandi si realizzano usando operazioni indirette. Qualsiasi istruzione macchina indiretta (come LDA (),Y o STA(),Y) deve essere evitata perchè risultano in stati bus "falsi" che vengono letti dall'8563 e di conseguenza eseguiti come se fossero istruzioni valide.
2. non dovete accedere, direttamente o indirettamente, all'8563 durante un'interruzione, perchè non si possono salvare o memorizzare i registri 8563 senza intralciare qualsiasi I/O in corso al momento dell'interruzione.

DETTAGLI PER LA PROGRAMMAZIONE DEL CHIP AD 80 COLONNE

Fino ad ora, avete imparato che il chip 8563 ha:

1. 37 registri interni
2. 16K di RAM indipendente (che non può essere indirizzata nella memoria del C128)
3. un registro indirizzo ($D600 e un registro dati $D601) nella memoria I/O del C128.

La Figura 11-1 è un riassunto dei registri dell'8563, in forma di mappa di registro.

REG	BIT 7	6	5	4	3	2	1	0	
0	HT7	HT6	HT5	HT4	HT3	HT2	HT1	HT0	Totale orizzontale
1	HD7	HD6	HD5	HD4	HD3	HD2	HD1	HD0	Orizzontale visualizzato
2	HP7	HP6	HP5	HP4	HP3	HP2	HP1	HP0	Posizione di sincronizzazione orizzontale
3	VW3	VW2	VW1	VW0	HW3	HW2	HW1	HW0	Ampiezza di sincr. Verticale/Orizzontale
4	VT7	VT6	VT5	VT4	VT3	VT2	VT1	VT0	Totale verticale
5	—	—	—	VA4	VA3	VA2	VA1	VA0	Regolazione Totale Verticale
6	VD7	VD6	VD5	VD4	VD3	VD2	VD1	VD0	Verticale Visualizzato
7	VP7	VP6	VP5	VP4	VP3	VP2	VP1	VP0	Posizione di Sincronizzazione Verticale
8	—	—	—	—	—	—	IM1	IM0	Modo di Intercalazione
9	—	—	—	CTV4	CTV3	CTV2	CTV1	CTV0	Carattere Totale Verticale
10	—	CM1	CM0	CS4	CS3	CS2	CS1	CS0	Modo Cursore, Inizio Scansione
11	—	—	—	CE4	CE3	CE2	CE1	CE0	Cursore Fine della Linea di Scansione
12	DS15	DS14	DS13	DS12	DS11	DS10	DS9	DS8	Indirizzo Inizio Video Alto
13	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0	Indirizzo Inizio Video Basso
14	CP15	CP14	CP13	CP12	CP11	CP10	CP9	CP8	Posizione del Cursore Alta
15	CP7	CP6	CP5	CP4	CP3	CP2	CP1	CP0	Posizione del Cursore Bassa
16	LPV7	LPV6	LPV5	LPV4	LPV3	LPV2	LPV1	LPV0	Penna Ottica Verticale
17	LPH7	LPH6	LPH5	LPH4	LPH3	LPH2	LPH1	LPH0	Penna Ottica Orizzontale
18	UA15	UA14	UA13	UA12	UA11	UA10	UA9	UA8	Indirizzo di Aggiornamento Alto
19	UA7	UA6	UA5	UA4	UA3	UA2	UA1	UA0	Indirizzo di Aggiornamento Basso
20	AA15	AA14	AA13	AA12	AA11	AA10	AA9	AA8	Inizio Attributo dell'Ind. Alto
21	AA7	AA6	AA5	AA4	AA3	AA2	AA1	AA0	Inizio attributo dell'Ind. Basso
22	CTH3	CTH2	CTH1	CTH0	CDH3	CDH2	CDH1	CDH0	Carattere Tot. (h), Vid. (V)
23	—	—	—	CDV4	CDV3	CDV2	CDV1	CDV0	Carattere Vid. (v)
24	COPY	RVS	CBRATE	VSS4	VSS3	VSS2	VSS1	VSS0	Scroll dolce verticale
25	TEXT	ATR	SEMI	DBL	HSS3	HSS2	HSS1	HSS0	Scroll dolce orizzontale
26	FG3	FG2	FG1	FG0	BG3	BG2	BG1	BG0	Colore di Primo Piano/di Fondo
27	AI7	AI6	AI5	AI4	AI3	AI2	AI1	AI0	Incremento dell'Indirizzo/Riga
28	CB15	CB14	CB13	RAM	—	—	—	—	Indirizzo di Base del Carattere
29	—	—	—	UL4	UL3	UL2	UL1	UL0	Linea di Scansione di sottolineatura
30	WC7	WC6	WC5	WC4	WC3	WC2	WC1	WC0	Conteggio delle Parole
31	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0	Dato
32	BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8	Indirizzo di Inizio Blocco Alto
33	BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0	Indirizzo di Inizio Blocco Basso
34	DEB7	DEB6	DEB5	DEB4	DEB3	DEB2	DEB1	DEB0	Inizio Abilitazione Video
35	DEE7	DEE6	DEE5	DEE4	DEE3	DEE2	DEE1	DEE0	Fine Abilitazione Video
36	—	—	—	—	DRR3	DRR2	DRR1	DRR0	Frequenza di quadro DRAM
37	HSy	NCV	SyNC	—	—	—	—	—	Polarità di Sincron, Vertic., Orizzon.

Figura 11-1. Mappa dei Registri VDC dell'8563

I numeri nella colonna a sinistra sono i numeri dei registri. Quando programmate l'8563, vi riferite ai registri solo attraverso i numeri, poichè non hanno un vero indirizzo nella memoria del C128. All'interno del grafico dei registri, le colonne da 7 a 0 si riferiscono ai bit all'interno dei registri. A destra ci sono i nomi dei registri per funzione. Molti dei registri controllano più di un'operazione del chip, così i nomi si riferiscono solo al primo scopo del registro. Questi registri vengono trattati individualmente alla fine di questo capitolo. Alcuni registri chiave vengono trattati nella sezione seguente. Per una spiegazione di ogni registro, vedete la descrizione registro per registro in fondo a questo capitolo.

I 16K della RAM dell'8563 sono organizzati dal Kernal nella seguente struttura di default:

TIPO DI MEMORIA	LOCAZIONE RAM 8563
Area Visualizzaz. Carattere	$0000-$07FF
Attributi del Carattere (colore)	$0800-$0FFF
Definizioni del Carattere	$2000-$3FFF

Il tracciato può essere modificato dal vostro programma di applicazione. I registri 12 e 13 (byte alto/byte basso) specificano l'inizio dell'area di visualizzazione del carattere, che si può comparare allo schermo della RAM nel chip VIC. I registri 20 e 21 (byte alto/byte basso) specificano l'indirizzo base degli attributi del carattere. Questi registri determinano l'indirizzo di partenza delle caratteristiche video di un carattere come video inverso, sottolineatura, lampeggiatore, colore. Il registro 28 determina l'indirizzo base del carattere, l'inizio delle definizioni del carattere nella RAM dell'8563.

Come potete vedere, il video 8563 ha le stesse tre componenti del video VIC: memoria dello schermo, memoria del colore e definizioni dei caratteri. Tutte e tre si trovano nei 16K della RAM dell'8563 e non appaiono nella memoria grafica del C128. Queste componenti del video dell'8563 sono trattate più in dettaglio nella prossima sezione.

Il terzo elemento della programmazione dell'8563 sono i registri indirizzo e dati che sono nelle locazioni $D600 e $D601. Queste locazioni sono le transizioni, le entrate ed uscite dal chip 8563. Non potete riferirvi direttamente ad un registro 8563 o ad una locazione RAM. Dovete leggere o scrivere indirettamente in $D600 e $D601 per comunicare tra il chip 8563 ed il resto del Commodore 128. La Figura 11-2 dà la descrizione di ognuno dei registri rilevati e dei bit nel loro interno:

$D600→indirizzo	**(scrivi)**	**:**	**—**	**—**	**R5**	**R4**	**R3**	**R2**	**R1**	**R0**
stato	**(leggi)**	**:**	**STATUS**	**LP**	**VBLANK**	**—**	**—**	**VER0**	**VER1**	**VER2**
$D601→dato	**(l/s)**	**:**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**

Figura 11-2. Registri Grafici

Alla locazione $D600, il bit 7 agisce come bit di stato. Quando richiedete un'operazione di scrittura o di lettura alla RAM 8563 o ad un registro, dovete controllare il valore del bit 7 del $D600. Se è 0, non può avvenire alcuna operazione di lettura o di scrittura. Questa è la sequenza di passi per eseguire un'operazione di lettura o di scrittura:

1. indirizzatevi al registro (per numero) o alla locazione interna RAM
2. verificate il valore del bit 7 finchè sia uguale ad 1 perchè il dato sia valido
3. una volta che il bit 7 è alto (1), eseguite l'operazione di lettura o di scrittura.

Più avanti in questo capitolo, vedrete degli esempi di programma che illustrano il comportamento di questi registri.

Il resto dei bit in $D600 tranne il bit 6, che coinvolge una penna ottica, sono usati per rappresentare un numero di registro per l'accesso di lettura o di scrittura. Sono necessari sei bit perchè l'8563 ha 37 registri; perciò, sono necessari abbastanza bit per rappresentare i numeri fino a 37.

La locazione $D601 è strettamente un registro dati che passa i valori tra l'8563 e l'8502 durante le operazioni di lettura e di scrittura rispettivamente. Ogni bit in questo registro è un data bit (bit di dato). Tutti i dati che passano tra l'8563 e l'8502 devono passare attraverso questo registro.

BACKGROUND FONDAMENTALE DEL CHIP 8563

CARATTERI

Come sapete, il chip 8563 è progettato in primo luogo per visualizzare i caratteri in una matrice di 80 per 25 caratteri. Il carattere visibile è una matrice di 8 per 8 caratteri. Il carattere può essere definito da 16 o 32 byte e possono essere visualizzati meno di 8 per 8 pixel.

L'unità di misura orizzontale all'interno di un carattere è un pixel. L'unità di misura verticale all'interno di un carattere è chiamata una scan line (linea di scansione). Una linea di scansione è una singola riga orizzontale di pixel esaminati che si chiama raster. Il pixel e la linea di scansione sono le più piccole unità di misura del chip 8563 nelle direzioni orizzontale e verticale, rispettivamente. La misura dell'immagine del carattere visualizzato sia nei pixel che nelle linee di scansione è programmabile ed è determinata da quattro registri nell'8563:

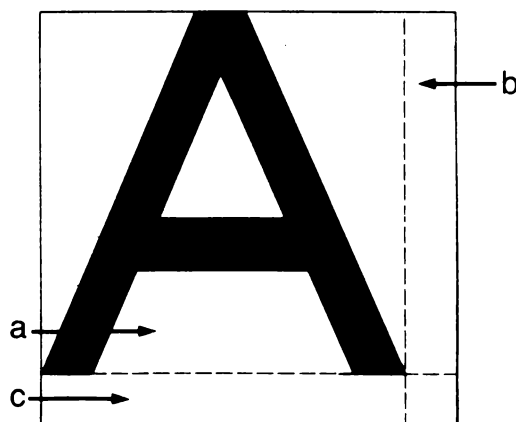



Figura 11-3

a. Carattere visibile di 8 per 8 pixel.
b. Spazio orizzontale tra caratteri.
c. Spazio verticale tra caratteri.

Carattere Totale (R9) (verticale), Carattere Totale (R22) (orizzontale), Carattere Visualizzato (R23) (verticale) e Carattere Visualizzato (R22) (orizzontale).

Questi registri sono illustrati nella descrizione registro per registro alla fine di questo capitolo.

RETICOLO (FRAME)

Un reticolo è un vettore a due dimensioni (matrice) di caratteri, ci si riferisce ad essa semplicemente come schermo. Il default della frame è lo schermo 80 colonne per 25 righe nel quale i caratteri vengono visualizzati. Ogni carattere nella frame ha un puntatore di carattere (un byte della RAM di visualizzazione) che dice all'8563 quale carattere visualizzare in ogni posizione del carattere all'interno della frame. Pensate alla frame come ad un dipinto incorniciato su un muro contenente un vettore a due dimensioni (matrice) di caratteri. Per esempio, una frame di cui si è specificato che sia 16 per 12 caratteri è riempita di caratteri come nella Figura 11-4.

La frame contiene inoltre il tracciato verticale e tutte le altre parti non visibili dello schermo. La frame è programmabile nella misura, come una finestra (window). La frame dello schermo determina la misura dell'immagine che si vede, come una cornice intorno ad una finestra.

COMPRENSIONE DEL VIDEO AD 80 COLONNE E MEMORIA DEGLI ATTRIBUTI

Come avete imparato precedentemente in questo capitolo, l'8563 ha tre aree di memoria che controllano il video ad 80 colonne: definizioni dei caratteri, attributi dei caratteri e memoria di visualizzazione dello schermo.

In una parte precedente, sono stati introdotti i tre elementi principali della programmazione del chip ad 80 colonne: (1) i registri indirizzo e dati $D600 e $D601 che dispongono nella mappa della memoria RAM del C128, (2) i 16K di RAM indipendente dell'8563, e (3) i 37 registri interni. I 16K di RAM indipendente dell'8563 forniscono la maggior parte degli elementi di informazione video: memoria video, attributi dei caratteri, definizioni dei caratteri. La memoria video è comparabile alla memoria video nel chip VIC. Gli attributi dei caratteri forniscono il colore ed altre informazioni per i caratteri e sono simili alla funzione colore della RAM del VIC. Le definizioni dei caratteri dell'8563 corrispondono al carattere ROM del VIC. Sebbene le memorie video del VIC e dell'8563 siano simili, le loro operazioni sono diverse. Questo fornisce una frame di riferimento alle caratteristiche del chip VIC, così potete capire il confronto tra le operazioni dei processori video sia del VIC e dell'8563.

Tutte queste componenti video sono memorizzate nei 16K della RAM indipendente dell'8563. Ancora, ecco la breakdown delle locazioni dove sono memorizzate le tre sezioni della memoria video della RAM 8563:

TIPO DI MEMORIA	**LOCAZIONE RAM 8563**
Memoria Video	**$0000-$07FF**
Attributi dei Caratteri	**$0800-$0FFF**
Definizioni dei Caratteri	**$2000-$3FFF**

Tutta la memoria video dell'8563 risiede nella sezione dei 16K di RAM indipendente. Le definizioni dei caratteri nella RAM 8563 richiedono fino a 32 byte per carattere. Se si usano 32 byte per carattere, le definizioni dei caratteri dell'8563 usano 8K di memoria per insieme di caratteri o un totale di 16K. Altrimenti, ogni insieme di caratteri occupa 4K della RAM 8563 indipendente, o un totale di 8K per entrambi gli insiemi di caratteri. Il chip VIC richiede un totale di 4K di ROM per rappresentare tutti i caratteri, poichè il chip VIC richiede solo 8 byte per rappresentare un carattere.

Il resto di questa parte descrive come l'area video, gli attributi dei caratteri e le definizioni dei caratteri sono ottenuti ed interpretati.

COMPRENSIONE DELLA MEMORIA VIDEO

La frame, che è stata introdotta in questo capitolo, appartiene alla memoria video, la parte di memoria che potete vedere sullo schermo. La memoria video risiede nell'intervallo $0000-$07FF nei 16K della RAM indipendente dell'8563. Pensate allo schermo come ad una frame di 25 righe per 80 caratteri l'una. Una frame può avere diverse proporzioni, ma qui considereremo la frame di 80 per 25.

Ogni posizione del carattere (locazione di memoria video) all'interno di una frame contiene un codice che agisce come un puntatore nelle definizioni del carattere, che dice all'8563 quale carattere visualizzare in quella determinata posizione all'interno della frame. Per lo schermo definito di frame 25 righe per 80 colonne, i primi 80 puntatori definiscono i codici dei caratteri per le 80 posizioni dei caratteri nella riga in alto dello schermo (frame). Dalla locazione 81 alla 160 definiscono quali caratteri sono visualizzati nella seconda riga dello schermo (frame), e così via. Per la frame dello schermo definita di 80 per 25, sono necessarie 2000 locazioni video per visualizzare le 2000 possibili posizioni del carattere nello schermo.

In modo testo, i contenuti della RAM video sono codici video come quelli che usa il chip VIC. Questi non sono codici ASCII CHR$. I codici video sono in realtà indici nelle definizioni della RAM dell'8563, come i codici video usati come indici nella ROM del VIC. Poichè il primo carattere nella RAM dell'8563 è il carattere "chiocciola", gli è assegnato il codice video 0. La A è il secondo carattere, così viene assegnato il codice video 1, e così via. Per una lista completa dei codici video, vedete l'Appendice D.

INDIRIZZAMENTO DELLA MEMORIA VIDEO

I contenuti dei registri 12 (byte alto) e 13 (byte basso) definiscono l'indirizzo di partenza dei 16 bit del primo carattere (in alto a sinistra) della frame (memoria video). Il default dell'indirizzo di partenza della memoria video è $0000. L'indirizzo della seconda posizione del carattere nella frame, cioè, il carattere nella colonna due, riga uno, è definito dai contenuti dei registri 12 e 13 più 1. Sommate un indice ai contenuti dei registri 12 e 13 per giungere all'indirizzo della posizione del carattere successivo della frame. Per esempio, per trovare l'indirizzo della posizione del carattere nella prima colonna della seconda riga di una frame 80 per 25, sommate un offset di 80 ai contenuti dei registri 12 e 13. Per trovare

l'indirizzo della posizione del carattere nella colonna 1, riga 3, sommate 160 ai contenuti dei registri 12 e 13. In generale, sommate un offset di 1 per ogni posizione del carattere a destra dell'angolo in alto a sinistra della frame. Per raggiungere la stessa colonna nella riga sotto alla posizione corrente del carattere, sommate un offset di 80. Sebbene lo schermo (frame) appaia come una matrice a due dimensioni di caratteri, i riferimenti dell'indirizzo nella memoria sono accessibili in sequenza lineare. Per esempio, sebbene il carattere nell'angolo in alto a sinistra dello schermo sia solo a 24 righe dal carattere nell'angolo in basso a sinistra dello schermo, in memoria, le locazioni di quei caratteri sono in realtà distanti 1920 (24 per 80) locazioni di memoria. Così una frame, la misura di default dello schermo (80 per 25) richiede 2000 locazioni di memoria per memorizzare i contenuti delle 2000 posizioni dei caratteri nella frame. I contenuti di ogni locazione video RAM (che si comportano come un indicatore indicizzato nelle definizioni dei caratteri) dicono all'8563 quale carattere, fuori dai 256 caratteri disponibili in ogni insieme di caratteri, porre a qualsiasi posizione particolare nella frame 80 per 25 caratteri.

ATTRIBUTI DEI CARATTERI

Gli attributi dei caratteri sono un insieme di locazioni della RAM dell'8563 che assegnano qualità video di arricchimento ai caratteri nella frame. Ad ogni posizione del carattere nella frame viene assegnato il proprio byte attributo, così una frame (schermo) di 80 colonne per 25 righe richiede 2000 byte attributi. Il default della locazione dei byte attributi per la frame 80 per 25 è memorizzato nell'intervallo da $0800 a $0FCF nei 16K di RAM indipendente dell'8563.

Gli attributi (ed i corrispondenti bit) che arricchiscono le qualità video dei caratteri nella frame sono i seguenti:

BIT	ATTRIBUTI
7	Insieme di Caratteri Alternati
6	Attributo di Inversione Video
5	Sottolineatura dell'Attributo
4	Lampeggio dell'Attributo
3	Rosso
2	Verde
1	Blu
0	Intensità

L'**INSIEME DI CARATTERI ALTERNATI** (bit 7), se alto (1), abilita 256 caratteri in più ad essere visualizzati all'interno della frame.

L'**ATTRIBUTO DI INVERSIONE VIDEO** (bit 6), se alto (1), scambia il colore di fondo con quello in primo piano per i caratteri corrispondenti nella memoria video. Osservate anche la descrizione registro per registro del chip 8563 per tutti i dettagli sul registro 24, il Registro di Schermo Inverso.

L'ATTRIBUTO DI SOTTOLINEATURA (bit 5), se alto (1), abilita la **LINEA DI SCANSIONE DELLA SOTTOLINEATURA** (come definita dai bit 0 fino a 4 del registro 29) nel colore di fondo per quel carattere all'interno della frame. La linea di scansione della sottolineatura può essere qualsiasi linea di scansione nel carattere come determinato dai quattro bit più bassi del registro 29.

Quando **L'ATTRIBUTO DI LAMPEGGIO** (bit 4) è uguale ad 1, il carattere associato a questo byte di attributo lampeggia. Il lampeggio si ottiene dal continuo scambio tra il colore di fondo e quello in primo piano. Il ritmo del lampeggio è determinato dal bit 5 del registro 24.

I bit di attributi da 3 a 0 determinano il colore di fondo per **R**, **V**, **B**, e **I** rispettivamente. Il carattere all'interno della memoria video associata appare col colore di fondo determinato dai bit R, V, B e I. Il colore di fondo per ogni cella del carattere può avere uno dei 16 colori. I colori sullo schermo possono essere più di rosso, verde e blu. I segnali RVBI che vengono da ogni input e la combinazione di questi segnali determinano il reale colore dello schermo. Ecco i colori disponibili ed i codici associati ai colori. Il colore di fondo dei caratteri è determinato dai bit da 0 a 3 del registro 26. Questi bit del registro 26 specificano il colore di fondo per tutti i caratteri dello schermo. Notate, tuttavia, che in linguaggio macchina questi codici sono uno meno dei codici in BASIC.

1 Nero	**9 Porpora scuro**
2 Bianco	**10 Giallo scuro**
3 Rosso scuro	**11 Rosso chiaro**
4 Celeste	**12 Azzurro scuro**
5 Porpora chiaro	**13 Grigio**
6 Verde scuro	**14 Verde chiaro**
7 Blu scuro	**15 Blu chiaro**
8 Giallo chiaro	**16 Grigio chiaro**

Codici dei Colori BASIC nell'Output ad 80 colonne.

INDIRIZZAMENTO DEGLI ATTRIBUTI DEI CARATTERI

I contenuti dei registri 20 (byte alto) e 21 (byte basso) definiscono l'indirizzo dell'inizio della memoria attributi. Il default dell'indirizzo iniziale della memoria attributi è $0800. I contenuti di questi due registri formano l'indirizzo a 16 bit che segna l'inizio della memoria attributi.

Indirizzate questi attributi come per l'indirizzamento della memoria video. Per indirizzare gli attributi successivi dei caratteri a destra del carattere nella posizione **HOME** (in alto a sinistra) della frame, sommate un offset di 1 ai contenuti dei registri 20 e 21 per ogni attributo del carattere a destra della posizione HOME. Per indirizzare l'attributo del carattere direttamente sotto alla posizione HOME (riga 2, colonna 1), sommate un offset di 80 ai contenuti dei registri 20 e 21. Per indirizzare l'attributo del carattere nell'angolo in basso a sinistra della frame 80 per 25, sommate un offset decimale di 1920 (24 per 80).

DISABILITAZIONE DEGLI ATTRIBUTI DEI CARATTERI

Molti programmi di applicazione per il chip 8563 non hanno bisogno dell'assegnazione degli attributi dei caratteri. L'8563 ha un bit che permette agli attributi di essere disabilitati, salvando una notevole quantità di RAM. Il bit 6 del registro 25 determina se gli attributi siano abilitati o no. Se il bit 6 è acceso (1), gli attributi sono abilitati: se il valore del bit 6 è 0, gli attributi sono disabilitati. Se gli attributi sono disabilitati, la porzione di RAM degli attributi dell'8563 può essere usata per altri scopi. Se gli attributi sono disabilitati, è disponibile solo un insieme di 256 caratteri, e gli attributi di sottolineatura, di lampeggio e di inversione video non sono disponibili. Inoltre, tutti i caratteri sono visualizzati con lo stesso colore di fondo come determinato dai bit da 4 a 7 del registro 26. Il colore di fondo è determinato ancora dai 4 bit più bassi (3-0) del registro 26, senza badare al fatto che gli attributi siano abilitati o no. Il margine dello schermo e gli spazi tra i caratteri verticali ed orizzontali assumono per default il colore dello sfondo.

La memoria video può contenere più locazioni di memoria ed attributi di quelli che possono essere visualizzati in una volta in una frame 80 per 25. Questa porzione di memoria video può essere visualizzata modificando i contenuti dei registri 12 e 13, l'indirizzo di Inizio Video e i registri 20 e 21, l'indirizzo di Inizio Attributi, per considerare un nuovo inizio di memoria video e memoria attributi. Lo schermo (frame) può scorrere verticalmente incrementando o decrementando i valori dell'indicatore della memoria video e degli indirizzi dell'inizio attributi. Ciò fornisce un modo per scorrere il testo verticalmente in avanti o indietro nella memoria video.

Oltre a fare lo scrolling verticale nella memoria video, l'8563 ha una funzione che permette di fare lo scrolling nella memoria video in direzione orizzontale. Entrambi questi argomenti sono trattati più avanti in questo capitolo.

DEFINIZIONI DEI CARATTERI

Le definizioni dei caratteri che risiedono nella RAM dell'8563 specificano le forme dei caratteri che vedete nello schermo ad 80 colonne. Le forme dei caratteri in memoria definiscono la formazione dei pixel di ogni membro dell'insieme dei caratteri visualizzato sullo schermo. Le definizioni dei caratteri possono essere ridefinite dal vostro programma di applicazione.

L'intervallo di memoria di default per le definizioni dei caratteri va dalla locazione $2000 alla $3FFF nella RAM 8563. Questo intervallo ha un totale di 8K (8192 byte). Questa memoria è sufficiente per rappresentare due insiemi di caratteri di 256 caratteri ciascuno. Il bit di attributo 7 di ogni byte di attributo seleziona ciascuno degli insiemi di caratteri per ogni posizione del carattere in una frame. I caratteri dell'8563 sono rappresentati sia da 16 che da 32 byte, dipende dal valore dei 5 bit più bassi (4-0) del registro 9. Se il valore dei 5 bit più bassi del registro 9 è minore di 16, ogni carattere viene definito da 16 byte nella memoria del carattere. Perciò ogni insieme di caratteri richiede un totale di 4096 byte di RAM dell'8563. Se il valore dei cinque bit più bassi del registro 9 è maggiore o uguale a 16, ogni carattere è definito da 32 byte di RAM dell'8563. In questo caso ogni insieme di caratteri occupa 8192 byte (8K) di memoria RAM dell'8563 ed i caratteri definiti vengono visualizzati usando più di 16 linee di scansione per ciascuno. Questo occupa gli interi 16K di RAM dell'8563 e non lascia spazio per gli attributi. Tuttavia, questi caratteri più alti vengono usati raramente, perchè i caratteri sono alti il doppio di quanto sono larghi e sembrano sproporzionati.

Il primo byte di ogni definizione del carattere in memoria rappresenta la linea di scansione orizzontale in alto del carattere visualizzato sullo schermo. Secondo il valore dei cinque bit più bassi del registro 9, un carattere avrà 16 o 32 linee di scansione, ognuna di 8 bit (pixel) di larghezza.

Sebbene le definizioni del carattere nella RAM dell'8563 siano definite da 16 o 32 byte da 8 bit accatastati uno sull'altro, il Bus dei Dati Video è di solo 8 bit, cosa che limita la reale immagine del carattere di 8 pixel orizzontali. La larghezza totale del carattere, incluso lo spazio orizzontale tra caratteri, può superare gli 8 pixel. Per dettagli vedete la definizione di carattere all'inizio del capitolo e la descrizione registro per registro alla fine di questo capitolo.

INDIRIZZAMENTO DELLE CONFIGURAZIONI DELL'INSIEME DEI CARATTERI

La locazione di default della definizione del carattere comincia all'indirizzo $2000 nella RAM dell'8563, e poichè la locazione di inizio è programmabile, potete cambiarla. La locazione di inizio dell'insieme delle definizioni dei caratteri è determinata dal bit 5 fino al 7 del registro 28. Questi sono i tre bit più significativi dell'indirizzo a 16 bit che indica l'inizio delle definizioni dei caratteri nella RAM dell'8563. I bit dal 4 allo 0 del registro 9 costituiscono i 5 bit meno significativi per l'indirizzo che indica l'inizio delle definizioni dei caratteri dell'8563. Tuttavia, se il valore dei bit da 4 a 0 nel registro 9 è maggiore o uguale a 16, solo i bit da 7 a 5 sono significativi, poichè i caratteri sono costituiti da 32 byte ciascuno e tutti i 16K di memoria RAM dell'8563 sono allocati per i due insiemi di caratteri, 8K per ogni insieme.

LETTURA DEI REGISTRI (E DELLA RAM) DELL'8563

Come sapete, il chip 8563 ha i propri 16K di RAM ed i propri insiemi di registri. Nessuno di questi appare nella mappa di memoria del Commodore 128 standard. Le sole locazioni di memoria che si riferiscono al chip 8563 nella memoria del Commodore 128 sono le locazioni $D600 e $D601. Questi sono i registri indirizzo e dati dell'8563, rispettivamente. Per scrivere o leggere la RAM dell'8563 o i registri, dovete passare i dati attraverso i registri indirizzo e dati. Questa parte spiega come leggere i registri dell'8563. La parte seguente illustra come scrivere sui registri e sulla RAM dell'8563.

LETTURA DI UN REGISTRO 8563

La lettura di un registro 8563 richiede cinque passi distinti di programmazione. Sebbene si possa farlo in BASIC, si raccomanda caldamente di programmare l'8563 in linguaggio macchina. Brevemente ecco i cinque passi di programmazione in linguaggio macchina, l'algoritmo, per leggere un registro 8563. Segue una spiegazione più dettagliata ed un programma per leggere un registro 8563.

1. Caricate (subito) il registro X con il numero del registro che volete leggere.
2. Memorizzate i contenuti del registro X (il numero del registro 8563) nel registro indirizzo ($D600) dell'8563.
3. Verificate il bit di stato, bit 7 di registro indirizzo $D600, finchè è alto (1).
4. Se il bit di stato è basso (0), tornate indietro e ricontrollatelo.
5. Altrimenti, caricate l'accumulatore con i contenuti del registro dati dell'8563 $D601. Memorizzate i contenuti del registro dati in una locazione RAM del C128 (una variabile) per un uso futuro e stampate il valore della variabile.

Poichè molti registri dell'8563 sono indicatori di indirizzo a 16 bit, si richiede una coppia di registri, un registro per il byte alto ed uno per il byte basso. In questo caso, incrementate semplicemente il registro X e ripetete il procedimento a 5 passi per leggere i contenuti del registro del byte basso. Il registro del byte alto viene per primo nell'ordine dei registri, perciò il primo che viene letto è il byte alto di una coppia di registri, incrementate il registro X e ripetete il procedimento per il byte basso. Quando l'8502 usa una coppia di registri per indicare un indirizzo a 16 bit, il byte alto è il byte basso più 1.

Ecco un programma che legge la coppia dei registri del byte alto (R20) e del byte basso (R21) che agiscono come un indicatore dell'inizio della memoria degli attributi nella RAM 8563. Il valore memorizzato nelle locazioni $FB e $FC ritorna al default dell'inizio (byte alto/basso) della memoria degli attributi. Questo indicatore può essere rilocalizzato in modo da indicare una parte della RAM 8563 dove voi definite la RAM video e le configurazioni dei caratteri.

ESEMPIO:

Lettura di un registro (indirizzo base dell'attributo)

```
10 SYS DEC("1800")
20 A$=HEX$(PEEK(251))
30 B$=HEX$(PEEK(252))
40 C$=RIGHT$(A$,2)
50 D$=RIGHT$(B$,2)
60 E$=C$+D$
70 F$="$"
80 G$=F$+E$
90 PRINT"INDIRIZZO DI PARTENZA = ";G$

READY.
```

```
MONITOR
    PC  SR AC XR YR SP
; FB000 00 00 00 00 F8

. 01800  A2 14     LDX #$14
. 01802  20 0F 18  JSR $180F
. 01805  85 FB     STA $FB
. 01807  E8        INX
. 01808  20 0F 18  JSR $180F
. 0180B  85 FC     STA $FC
. 0180D  60        RTS
. 0180E  00        BRK
. 0180F  8E 00 D6  STX $D600
. 01812  2C 00 D6  BIT $D600
. 01815  10 FB     BPL $1812
. 01817  AD 01 D6  LDA $D601
. 0181A  60        RTS
```

Ecco i dettagli per ogni istruzione.

- La prima istruzione ($1800) carica il registro X con il numero di registro 20 ($14), il byte alto con indirizzo di inizio dell'attributo.
- La seconda istruzione salta alla subroutine $180F.
- La prima istruzione della subroutine memorizza i contenuti del registro X, ($14), nel registro indirizzo dell'8563 (locazione $D600 nella memoria I/O del C128).
- La seconda istruzione della subroutine (BIT $D600) collauda il bit di stato (7) **UPDATE READY** del registro di indirizzo dell'8563. Ciò è necessario poichè un'operazione di lettura o di scrittura non inizia nel momento in cui quelle istruzioni vengono emesse. Il bit Update Ready (bit 7) del registro Indirizzo dell'8563 decide quando l'istruzione di lettura (o scrittura) può essere eseguita. L'operazione di lettura è in sospeso secondo il valore del bit 7 del registro indirizzo. Mentre l'operazione di lettura è in sospeso (in prelatch), il valore del bit 7 è 0. Questo indica che il dato dell'operazione di lettura (o di scrittura) posto nel registro 31, non è ancora valido. Solo quando il valore del bit 7 nel registro di indirizzo è uguale a 1, il dato nel registro dei dati ($D601) è valido.

Il comportamento del chip 8563 ed il modo in cui i dati sono trasferiti tra i registri dell'8563, gli indirizzi ed i registri dei dati suggerisce l'uso delle istruzioni BIT. La natura dell'istruzione BIT 8502 è fatta apposta per controllare lo stato del bit Update Ready della locazione $D600. Quando il microprocessore incontra l'istruzione BIT, il valore del bit 7 è trasferito nel bit flag negativo del registro dell'8502. Se il valore è uguale a 0, la condizione negativa non esiste. Se il valore è 1, esiste una condizione negativa. Secondo il valore di questo bit, potete trasferire il controllo ad istruzioni che eseguono un'operazione secondo il valore di questo flag. Ecco dove entra in gioco la terza istruzione della subroutine ($1815).

- L'istruzione 3 (BPL $1812) è un'istruzione condizionata che si trasferisce a seconda del valore del bit flag negativo nel registro 8502.
  In questo programma, l'istruzione BPL (Branch on Result Plus) salta indietro

fino alla locazione $1812 ed esegue l'istruzione BIT mentre il bit flag negativo è uguale a 0. L'istruzione BIT viene eseguita ancora, ed il valore del bit 7 è ancora trasferito al bit flag negativo nel registro 8502. Poi l'istruzione BPL viene incontrata una seconda volta nel ciclo. Se il valore del bit di stato è ancora 0, il trasferimento viene effettuato ancora. Il ciclo continua finchè il bit di stato negativo è 1. Solo quando il bit di stato è 1 il ciclo va alle istruzioni che seguono immediatamente le istruzioni di trasferimento.
Prima che il ciclo vada alle istruzioni dopo il trasferimento, il bit di stato Update Ready diventa un 1, è trasferito al bit flag negativo nel registro 8502, il trasferimento è saltato ed il dato può essere letto (caricato) con la quarta istruzione della subroutine ($1817) in $D601.
Solo dopo che il bit di stato diventa alto (1) è valido il dato in $D601.
Questo è il punto in cui il dato viene comunicato dal chip 8563 alla memoria del Commodore 128. Ora potete leggere ed operare sui contenuti dell'indirizzo all'interno del registro $14 del chip 8563.

- La quarta istruzione carica il valore memorizzato nel registro dati dell'8563 ($D601) nell'accumulatore.
- La terza istruzione del programma principale memorizza il valore del registro dati nella locazione $FB per un uso futuro. Successive operazioni di lettura da ($D601) distruggono il valore originale, per questo conviene sempre memorizzare i contenuti di $D601 in una variabile o in una locazione libera della RAM.
- L'istruzione successiva, INX, incrementa i contenuti del registro X, che seleziona il numero di registro successivo, $15, così il byte basso dell'indirizzo a 16 bit dell'inizio della memoria degli attributi può essere letto.
- Il resto del programma compie la stessa sequenza di passi descritti sopra per il byte basso dell'inizio della memoria degli attributi tranne che per l'istruzione RTS. L'istruzione RTS riporta il controllo del programma al programma principale.

La parte BASIC del programma stampa i byte alti e bassi dell'indirizzo a 16 bit che definisce l'inizio della memoria degli attributi.

Per trovare l'indirizzo reale della memoria degli attributi nella RAM 8563, ponete insieme il byte alto e quello basso, ed il numero esadecimale di quattro cifre (four-digit) definisce l'inizio della memoria degli attributi. Dentro il monitor, immettete questo comando di memoria:

```
M FB
```

I contenuti di $FB e di $FC (byte alto/basso) segnano l'inizio della memoria degli attributi.

SCRITTURA SU UN REGISTRO 8563

L'algoritmo di 5 passi per leggere i dati da un registro 8563 è simile a quello per scrivere su un registro 8563. Nell'ultima parte, il Passo 5 nell'algoritmo di lettura caricava l'accumulatore con il valore contenuto nel registro dati ($D601) dell'8563. Quando si scrive, esso compie l'operazione opposta. Invece di caricare l'accumulatore come per l'operazione di lettura, memorizza il valore dell'accumulatore nel registro dati.

Ecco un algoritmo per scrivere su un registro 8563:

1. Caricate il registro X con il numero di registro nel quale volete scrivere.
2. Caricate l'accumulatore con un valore che state scrivendo sul registro 8563.
3. Memorizzate i contenuti del registro X nel registro indirizzo dell'8563 in $D600.
4. Verificate il bit 7, il bit di stato, del Registro Indirizzo finchè sia alto (1).
5. Se lo stato è basso (0), rifate il ciclo e verificatelo ancora finchèè alto.
6. Quando il bit di stato (7) è alto, memorizzate il valore contenuto nell'accumulatore nel registro dati ($D601) dell'8563.

Ecco un programma campione che scrive sulla coppia di registri R18 e R19, la locazione Update della RAM. La coppia di registri R18 e R19 è la via che porta alla RAM indipendente dell'8563. Per ora illustreremo come scrivere su un registro 8563. Più avanti in questo capitolo, faremo un passo successivo per spiegare come leggere e scrivere sui 16K di RAM indipendente dell'8563.

```
READY.

MONITOR
     PC   SR AC XR YR SP
; FB000  00 00 00 00 F8

. 01800   A2 12      LDX #$12
. 01802   A9 20      LDA #$20
. 01804   20 1E 18  JSR $181E
. 01807   E8         INX
. 01808   A9 00      LDA #$00
. 0180A   20 1E 18  JSR $181E
. 0180D   EA         NOP
. 0180E   A2 12      LDX #$12
. 01810   20 2C 18  JSR $182C
. 01813   85 FB      STA $FB
. 01815   E8         INX
. 01816   20 2C 18  JSR $182C
. 01819   85 FC      STA $FC
. 0181B   60         RTS
. 0181C   00         BRK
. 0181D   00         BRK
```

```
. 0181E  8E 00 D6 STX $D600
. 01821  2C 00 D6 BIT $D600
. 01824  10 FB    BPL $1821
. 01826  8D 01 D6 STA $D601
. 01829  60       RTS
. 0182A  EA       NOP
. 0182B  EA       NOP
. 0182C  8E 00 D6 STX $D600
. 0182F  2C 00 D6 BIT $D600
. 01832  10 FB    BPL $182F
. 01834  AD 01 D6 LDA $D601
. 01837  60       RTS
```

- La prima istruzione (in $1800) carica il registro X con il numero di registro 18 ($12).
- L'istruzione successiva carica l'accumulatore con il byte alto del valore che state ponendo nel registro 18 ($20).
- L'istruzione 3 chiama la subroutine alla locazione $181E, che essenzialmente è la stessa routine dell'esempio del registro di lettura nell'ultima sezione, con un'eccezione. L'istruzione nella subroutine immediatamente prima di RTS è un'istruzione di memorizzazione nel registro dati $D601 dell'8563. Le due istruzioni ad essa precedenti formano un ciclo che controlla continuamente il bit di stato Update Ready fino all'operazione di scrittura, proprio come faceva il programma di lettura.
  L'operazione di scrittura (STA $D601) non viene compiuta finchè il bit di stato 7 nel registro indirizzo diventa alto (1). Una volta che il bit di stato diventa 1, il byte alto dell'indirizzo nel registro 18 viene scritto nel registro dati ($D601). La subroutine riporta il controllo alla parte principale del programma con l'istruzione RTS.
- Il programma riprende con la quarta istruzione INX. Questa incrementa il numero del registro.
- La quinta istruzione (LDA#00) carica il valore del byte basso dell'indirizzo perchè venga scritto nel registro 19 ($13) dell'accumulatore.
- La subroutine in $181E viene richiamata e lo stesso procedimento è ripetuto.
- Al ritorno della subroutine, viene compiuto lo stesso esempio di routine di lettura dell'ultima sezione (la subroutine tra $182C-$1837) per assicurarsi che siano validi i valori che sono stati scritti nei registri 18 e 19. Questi valori vengono stampati dalla routine BASIC. Per vedere quei valori, usate il comando di memoria all'interno del monitor come segue:

  M FB

  I primi due byte visualizzati sono:

  20 00

Questo mostra che l'indirizzo nella locazione update della coppia di registri (R18, R19) è stato scritto con successo ed è valido.

SCRITTURA SULLA RAM 8563

Ora prendete l'esempio dalla precedente sezione un passo più avanti. L'esempio precedente illustrava come scrivere su un registro dell'8563. Il registro scelto per l'esempio è la coppia di registri della locazione update (R18, R19). Questa è la coppia di registri che vi permette di scrivere sui e di leggere dai 16K della RAM indipendente dell'8563. Questa parte estende l'esempio precedente e vi mostra come scrivere sui 16K di RAM 8563.

Il programma sottostante riempie di caratteri la memoria video nella RAM 8563, nel procedimento che visualizza i caratteri sullo schermo. Il programma riempie il video RAM localizzato nel default d'intervallo $0000 fino a $07 CF, con il codice di carattere dello schermo 102 ($66). Vedete l'Appendice D per una lista completa dei codici dei caratteri dello schermo.

Ecco il listato:

```
READY.

MONITOR
    PC   SR AC XR YR SP
; FB000 00 00 00 00 F8

. 01800  A2 12     LDX #$12
. 01802  A9 00     LDA #$00
. 01804  20 21 18  JSR $1821
. 01807  E8        INX
. 01808  20 21 18  JSR $1821
. 0180B  A0 00     LDY #$00
. 0180D  A9 08     LDA #$08
. 0180F  85 FA     STA $FA
. 01811  A9 66     LDA #$66
. 01813  20 1F 18  JSR $181F
. 01816  C8        INY
. 01817  D0 FA     BNE $1813
. 01819  C6 FA     DEC $FA
. 0181B  D0 F6     BNE $1813
. 0181D  00        BRK
. 0181E  00        BRK
. 0181F  A2 1F     LDX #$1F
. 01821  8E 00 D6  STX $D600
. 01824  2C 00 D6  BIT $D600
. 01827  10 FB     BPL $1824
. 01829  8D 01 D6  STA $D601
. 0182C  60        RTS
```

Per eseguire questo programma, date il comando Go (G) nel Monitor di Linguaggio Macchina. Perchè questo programma operi, la memoria I/O deve essere nel contesto della configurazione del monitor attualmente selezionato. La memoria I/O è sempre nel contesto della configurazione 15 ($F), così date questo comando per eseguire il programma:

G F1800

Le prime 5 istruzioni (memoria $1800-$1808) e la subroutine memorizzata nelle locazioni di memoria $1821 fino a $182C sono le stesse istruzioni contenute nel programma che ha scritto sui registri 18 e 19 nella parte precedente. Tuttavia, invece di porre l'indirizzo $2000 nella coppia di registri 18 e 19 come nella parte precedente, l'indirizzo $0000 viene posto là per segnare il default della locazione iniziale della memoria video della RAM 8563. Qui il programma comincia a porre il codice 102 dello schermo nella RAM video. Tutti i 2000 byte (8 pagine di memoria meno 48 byte) della RAM video sono riempiti con questo codice del carattere dello schermo.

Ecco come funziona il resto del programma. L'istruzione memorizzata nella locazione $180B, LDY #$00, inizializza il registro Y a 0. L'istruzione successiva, LDA #$08 carica l'accumulatore con il valore 8. Poi, STA $FA, memorizza i contenuti dell'accumulatore, che inizializza questa locazione a 8. La locazione $FA viene usata più tardi nel programma come contatore per il numero di pagine da riempire con il codice dello schermo 102. L'istruzione memorizzata in $1811, LDA #$66, carica l'accumulatore con il valore esadecimale ($66) del codice decimale 102 dello schermo.

L'istruzione che parte alla locazione $1813, JSR $181F, chiama la subroutine localizzata all'indirizzo $181F. La subroutine è il punto principale del programma e scrive realmente i dati nella memoria dello schermo. Questa è essenzialmente la stessa routine della subroutine localizzata tra $181E e $1829, quella della parte precedente. Tuttavia, c'è una differenza basilare. La prima istruzione della subroutine memorizzata che comincia a $181F, LDX #$1F, carica il registro X con il numero di registro 31 ($1F). Quando scrivete sulla RAM 8563, dovete indirizzare il registro 31 perchè agisce come registro dati interno del chip 8563. Tutti i dati che saranno scritti nella RAM 8563 devono passare dal registro 31.

Le tre istruzioni successive:

```
STX $D600
BIT $D600
BPL $1824
```

verificano il bit di stato (7) del registro indirizzo finchè il valore diventa 1 (alto). Queste sono le stesse tre istruzioni usate nelle ultime due parti per la lettura o la scrittura in un registro o nella prima subroutine di questo programma.

Una volta che il bit di stato del registro indirizzo dell'8563 diventa alto, il dato che risiede nell'accumulatore (in questo caso) è scritto nel registro dati esterno dell'8563 all'indirizzo $D601, come nell'istruzione memorizzata alla locazione $1829. Inoltre, il dato nell'accumulatore è scritto nell'indirizzo contenuto nella coppia di registri update (R18, R19). Ciò accade perchè avete indirizzato il registro dati interno 31 ponendo il valore 31 nel registro indirizzo dell'8563 in $D600. Quando il bit di stato è alto, il dato nel registro 31 è posto nell'indirizzo corrente della RAM 8563 specificata dalla coppia di registri update (R18, R19). Avete posto questo indirizzo nella coppia di registri update all'inizio del programma.

A questo punto, l'hardware dell'8563 assume il controllo. Esso incrementa automaticamente l'indirizzo contenuto nella coppia di registri update (R18, R19). Voi in qualità di programmatori non incrementate i contenuti della coppia di registri update; il chip 8563 lo fa per voi automaticamente. Potete dire che la

coppia di registri update si stia autoincrementando. Inoltre, l'8563 legge i dati dall'indirizzo incrementato all'interno della RAM 8563, e pone i contenuti nel registro interno 31 dell'8563. Questo salva anche il lavoro di programmazione. Ricordate, entrambi questi compiti sono svolti dall'hardware dell'8563, non dal programmatore.

Poichè la coppia di registri update incrementa automaticamente gli indirizzi della RAM dell'8563, tutto quello che dovete fare è chiamare continuamente la subroutine che scrive i dati in ogni locazione successiva della RAM video. Questo è ciò che le istruzioni tra gli indirizzi $1813 e $181B fanno nel programma. Queste istruzioni chiamano la subroutine per otto pagine della RAM video, o per un totale di 2048 volte. La subroutine di scrittura scrive solo un carattere per chiamata sullo schermo, così per riempire l'intero schermo, deve essere chiamata 2000 (80 per 25) volte.

Questo campione di programma in BASIC mostra come leggere e scrivere opportunamente sui registri dell'8563, e come leggere, modificare e scrivere in qualsiasi locazione della RAM 8563. Le due subroutine ML sono le chiavi e possono essere usate sia dal BASIC come mostrato qui, o dal vostro programma ML.

```
100 VI=DEC("1800"):VO=DEC("180C"):BANK15
110 READ A$
115 DO UNTIL A$="END"
116 :  POKE VI+I,DEC(A$)
117 :  I=I+1
118 :  READ A$
119 :  LOOP
120 :  DATA 8E,00,D6,2C,00,D6,10,FB,AD,01,D6,60
130 :  DATA 8E,00,D6,2C,00,D6,10,FB,8D,01,D6,60,END
140 DO
150 :  PRINT
160 :  INPUT"INDIRIZZO (0-16383)";AD
170 :  SYS VO, AD/256  ,18
180 :  SYS VO, ADAND255,19
190 :  SYS VI,,31
200 :  RREG A
210 :  PRINT"ALLA LOCAZIONE";AD;"LA MEMORIA E'";A
220 :  INPUT"NUOVO DATO (0-255)";A
230 :  SYS VO, AD/256  ,18
240 :  SYS VO, ADAND255,19
250 :  SYS VO,,31
260 LOOP
```

Il seguente disassemblaggio mostra le due routine ML usate nel programma BASIC precedente e negli esempi di registro di Lettura, registro di Scrittura, e Scrittura nella RAM nella sezione precedente. Essi sono rilocalizzabili.

```
01800  8E 00 D6  STX $D600
01803  2C 00 D6  BIT $D600
01806  10 FB     BPL $1803
01808  AD 01 D6  LDA $D601
0180B  60        RTS

0180C  8E 00 D6  STX $D600
0180F  2C 00 D6  BIT $D600
01812  10 FB     BPL $180F
01814  8D 01 D6  STA $D601
01817  60        RTS
```

SCRITTURA DI UN BLOCCO E COPIA DI UN BLOCCO

Le funzioni Scrittura Blocco e Copia Blocco del chip 8563 sono estensioni delle operazioni di scrittura descritte nella sezione precedente. La funzione Scrittura Blocco scrive lo stesso dato contenuto nell'accumulatore in un numero specificato di locazioni di memoria successive dell'8563. Ecco la procedura:

1. Ponete l'indirizzo iniziale, dove comincia la scrittura sulla RAM, nei registri 18 (byte alto) e 19 (byte basso).
2. Poi, caricate l'accumulatore con il dato che volete porre nelle locazioni successive della RAM 8563.
3. Caricate il registro X con il numero 31 che specifica il registro dati nell'8563.
4. Aspettate che il bit di stato Update Ready sia alto (1). Fino ad ora è la stessa procedura per la scrittura nella RAM 8563 che è stata descritta nella sezione Scrittura sulla RAM 8563.
5. Per selezionare la funzione Scrittura Blocco, azzerate (0) il bit 7 del registro 24, usando lo stesso procedimento per la scrittura su un registro.
6. Infine, ponete il numero delle locazioni di memoria successive che volete scrivere nella RAM 8563 nel registro 30.

A questo punto, l'8563 scrive il dato nel registro 31 cominciando dalla locazione determinata dai registri 18 e 19 fino al numero delle locazioni di memoria successive specificate dal registro 30. I contenuti dei registri 18 e 19 vengono incrementati automaticamente perchè si possa scrivere nella locazione seguente. Al completamento della Scrittura Blocco, i registri 18 e 19 contengono l'ultimo indirizzo più una delle locazioni della RAM 8563 nella quale avevate scritto.

Notate che un ciclo di scrittura segue l'operazione iniziale di scrittura nel registro 31; perciò, la quantità scritta nel registro 30 dovrebbe essere uno in meno del numero delle locazioni di memoria dell'8563 in cui volete scrivere.

COPIA DI UN BLOCCO

La funzione Copia Blocco copia (legge e scrive) i dati da una sezione della RAM 8563 ad un'altra. Per copiare un blocco della RAM 8563:

1. Ponete l'indirizzo di destinazione, dove comincia la copiatura nella RAM, nei registri 18 (byte alto) e 19 (byte basso).
2. Aspettate che il bit Update Status Ready divenga alto (1).
3. Ponete (1) il bit 7 del registro 24, che seleziona la funzione Copia Blocco. A questo punto, l'8563 scrive l'indirizzo della fonte iniziale per la copia del blocco nei registri 32 (byte alto) e 33 (byte basso).
4. Ponete il numero delle successive locazioni di memoria da copiare nel registro 30.

L'8563 legge i contenuti del primo indirizzo di origine e scrive il dato nel primo indirizzo di destinazione. Gli indirizzi della RAM 8563 vengono incrementati automaticamente e la copiatura continua finchè il numero delle locazioni di memoria specificate dal registro 30 sono copiate.

LETTURA DALLA RAM 8563

Questo procedimento di lettura dalla RAM 8563 è lo stesso della lettura da un registro, tranne che voi leggete dal registro dati, il registro 31.

PROGRAMMAZIONE DEL CURSORE 8563

Il cursore ad 80 colonne è programmabile per immagine e movimento. Primo, l'immagine del cursore può essere programmata in quattro modi diversi. I bit 6 e 5 del registro 10 controllano le quattro caratteristiche seguenti del cursore:

VALORE DEL BIT		CARATTERISTICA
6	5	
0	0	cursore fisso
0	1	cursore non visibile
1	0	lampeggio lento del cursore (1/16 della velocità della frame)
1	1	lampeggio veloce del cursore (1/32 della velocità della frame)

I bit dal 4 allo 0 del registro 10 determinano la linea di scansione in basso dove comincia la definizione del cursore all'interno delle 8 linee di scansione del blocco di caratteri del cursore. La linea di scansione 0 è in cima al blocco di caratteri del cursore, mentre la linea di scansione 8 è in fondo.

L'editor di interruzione comandata dello schermo ha una locazione indiretta che controlla il modo di programmazione del cursore. La locazione $0A2B nella RAM del C128 è il registro non indirizzabile che controlla i diversi modi per programmare il cursore dell'8563.

I bit dal 4 allo 0 del registro 11 specificano la linea di scansione più in alto dove la definizione del cursore termina entro le 8 linee di scansione del blocco di caratteri del cursore. I bit dal 4 allo 0 nei registri 10 e 11 specificano l'altezza del cursore. Questi registri vi permettono di programmare il cursore come un cursore di blocco fisso (per default) o come un cursore sottolineato.

POSIZIONAMENTO DEL CURSORE

I registri 14 (byte alto) e 15 (byte basso) determinano la posizione del cursore all'interno della frame visualizzata (schermo). Questa coppia di registri indica l'indirizzo del cursore all'interno della memoria video dell'8563. L'indirizzo del cursore può non essere uno nell'area video della frame corrente. Se questo indirizzo è all'interno della fascia di indirizzo della frame corrente (schermo), la posizione del carattere dove è visualizzato il cursore appare nel lampeggiamento del video inverso, come per la maggior parte dei cursori.

L'editor dello schermo di interruzione comandata ha anche una locazione indiretta che controlla il posizionamento del cursore sulla colonna e sulla riga. La locazione $0A35 nella RAM del C128 è il registro non indirizzabile per la selezione della colonna e della riga del cursore dell'8563.

BIT-MAP DEL VIDEO AD 80 COLONNE

Il chip 8563 fornisce un modo grafico (bit-map). La risoluzione dello schermo bit-map ad 80 colonne è di 640 per 400 pixel. L'8563 indirizza i byte che compongono il bit map in tutta la larghezza dello schermo, invece che nelle celle 8 per 8 dei caratteri come fa il chip VIC. Il modo bit-map viene selezionato posizionando il bit 7 del registro 25. Il Kernal inizializza il bit 7 a 0 per selezionare il modo testo. Le tre componenti principali di un video bit-map 8563, i dati, la memoria video ed il colore, sono simili al chip VIC per certe cose ma diverse per altre. Per esempio, il VIC ha una RAM bit-map di 8000 byte, nella quale ad ogni bit viene assegnato un singolo pixel sullo schermo visivo. Poichè il bit map 8563 ha una risoluzione doppia rispetto allo schermo VIC, usa il doppio della memoria: 16.000 byte (un pò meno di 16K). La corrispondenza uno a uno tra bit in memoria e pixel sullo schermo è simile, ma sono indirizzati diversamente. Gli indirizzi per accedere a quei bit sono tracciati in modo diverso in memoria. La Figura 9-25 del terzo Volume Capitolo 9 illustra come il bit map del VIC venga memorizzato. Lo schermo VIC memorizza 8 byte del bit-map negli 8 byte che normalmente creerebbero un carattere in modo carattere. Per esempio, i primi 8 byte del bit-map visibile sono memorizzati nella posizione del primo carattere (HOME), che porta automaticamente da 8192 ($2000) a 8199 ($2007), o dei primi 8 pixel delle prime otto righe del raster. Il carattere alla estrema destra della posizione HOME memorizza i secondi 8 pixel delle prime 8 righe del raster visibile. Le 200 righe del raster visibile dello schermo VIC sono larghe 320 pixel. Le 200 righe del raster visibile dell'8563 sono larghe 640 pixel. Sia lo schermo VIC che quello dell'8563 hanno righe del raster che non sono visibili sullo schermo. Questa parte è conosciuta come tracciato verticale. Lo schermo dell'8563 è memorizzato byte per byte lungo una riga del raster. I primi 8 pixel della prima riga del raster sono memorizzati nel primo byte del byte bit-map dell'8563, all'indirizzo $0000 della RAM 8563. I secondi 8 pixel della prima riga sono memorizzati nel secondo byte del bit-map ($0001 nella RAM 8563). I terzi 8 pixel sono memorizzati alla locazione $0002 e così via. La prima riga del raster è memorizzata nelle prime 80 locazioni di memoria nella RAM 8563, la seconda riga del raster è memorizzata nei secondi 80 indirizzi (80-159) della RAM 8563 e così via. Osservate la Figura 11-5.

	COLONNE DI CARATTERI DELL'8563												
RIGA DEL RA-STER	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	...	**80**
1	$0000	$0001	$0003	$0004	$0005	$0006	$0007	$0009	$000A	$000B	$000C	...	$004F
2	$0050	$0051	$0052	$0053	$0054	$0055	$0056	$0057	...	...	...	...	$009F
3	$00A0	...	...	...	...	...	...	...	...	...	...	...	...
4	$00F0	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...
25	$3E30	...	...	...	...	...	...	...	...	...	...	...	$3E7F

Figura 11-5. Rapporto tra i Dati Bit Map dell'8563 ed i Pixel dello Schermo

L'insieme dei 640 per 200 pixel sullo schermo formano l'immagine in questo modo: i bit che sono accesi (1) abilitano i pixel dello schermo con il colore di primo piano ed i bit spenti (0) assumono il colore di fondo.

PUNTATORE DEL BIT MAP

Il puntatore dell'inizio del bit-map dell'8563 porta automaticamente all'inizio della memoria video alla locazione $0000 della RAM 8563. L'indirizzo di partenza della memoria video è contenuto nei registri 12 (byte alto) e 13 (byte basso). I contenuti di questo indirizzo segnano l'indirizzo dei primi otto pixel della riga in alto del raster dello schermo (frame) 8563. Aggiungete un indice per accedere ai pixel successivi nelle righe successive del raster. Poichè l'intero bit map 640 per 200 usa quasi tutti i 16K disponibili nella RAM 8563, non c'è posto disponibile in memoria per gli attributi come il colore.

ASSEGNAZIONI DEL COLORE

Le assegnazioni del colore ai pixel per il bit map dell'8563 sono basate sul valore del bit in memoria che corrisponde al pixel sullo schermo. Se il valore del bit in memoria è uguale a zero, il pixel viene colorato con il colore di fondo secondo il valore dei bit da 0 a 3 nel registro 26 dell'8563. Se il valore del bit in memoria corrispondente al pixel dello schermo è uguale a uno, al pixel viene assegnato il colore di fondo determinato dai bit da 4 a 7 del registro 26. L'intero bit map di 640 per 200 pixel occupa quasi tutti i 16K della RAM 8563. Vi chiederete come fare ad usare gli attributi per le corrispondenti locazioni dello schermo. Avete due scelte:

o disabilitare gli attributi o ridurre la larghezza del vostro bit map di 2000 byte, e poi definire ed abilitare gli attributi. Con un bit map completo, non potete implementare gli attributi, semplicemente perchè non c'è abbastanza RAM 8563 indipendente per fare entrambe le cose. Se gli attributi sono disabilitati, lo schermo bit map dell'8563 usa solo due colori, come specificato dai codici dei colori di fondo e di primo piano nel registro 26. Se decidete di ridurre la larghezza del bit map dell'8563 e volete implementare gli attributi, ponete il bit 6 del registro 25 per abilitare gli attributi. Poi ponete gli indirizzi del byte alto e del byte basso nei registri 20 e 21, rispettivamente, per definire l'inizio della memoria degli attributi. Vedete la sezione degli Attributi dei Caratteri per ulteriori informazioni sugli attributi.

L'AMPIEZZA DELLA FRAME BIT MAP

L'ampiezza della frame bit-map porta direttamente alle dimensioni 640 per 200 delle righe del raster. Questa ampiezza della frame (schermo) può essere ridotta. Controllate la Descrizione Registro per Registro per dettagli sull'alterazione della misura dell'immagine bit-map visibile. La sezione successiva fornisce un esempio di un programma bit-map in linguaggio macchina che contiene un listato di programma e dettagli sulle routine importanti.

PROGRAMMA BIT MAP DELL'8563

Come sapete, l'8563 sostiene il modo bit map; tuttavia, i comandi grafici in BASIC non vengono visualizzati sullo schermo dell'8563. Questa sezione contiene un programma che trasferisce lo schermo VIC a 40 colonne allo schermo dell'8563. Lo schermo è ampio ancora 40 colonne, sebbene fornisca il modo per capire come è indirizzato lo schermo bit map dell'8563. Per una programmazione più facile, molte funzioni grafiche e in generale i progetti funzionali usano il chip VIC quando fanno il bit map di grafici ad alta risoluzione. Il chip VIC è progettato principalmente per questi scopi. L'8563 è progettato principalmente per la capacità video di 80 colonne. Ecco il listato del programma come appare nel Monitor di Linguaggio Macchina. Caricate il Monitor e cominciate ad inserire il programma all'indirizzo $1800.

```
. 01800  20 B3 18 JSR $18B3
. 01803  A2 19    LDX #$19
. 01805  A9 80    LDA #$80
. 01807  20 77 18 JSR $1877
. 0180A  A9 20    LDA #$20
. 0180C  85 FB    STA $FB
. 0180E  A9 00    LDA #$00
. 01810  85 FA    STA $FA
. 01812  A2 12    LDX #$12
. 01814  20 77 18 JSR $1877
. 01817  E8       INX
. 01818  20 77 18 JSR $1877
. 0181B  A9 00    LDA #$00
. 0181D  85 B1    STA $B1
```

```
. 0181F  A9 19     LDA #$19
. 01821  85 9B     STA $9B
. 01823  A9 07     LDA #$07
. 01825  85 9C     STA $9C
. 01827  A9 27     LDA #$27
. 01829  85 FE     STA $FE
. 0182B  EA        NOP
. 0182C  EA        NOP
. 0182D  A2 00     LDX #$00
. 0182F  A1 FA     LDA ($FA,X)
. 01831  20 75 18  JSR $1875
. 01834  20 83 18  JSR $1883
. 01837  EA        NOP
. 01838  EA        NOP
. 01839  EA        NOP
. 0183A  C6 FE     DEC $FE
. 0183C  D0 EF     BNE $182D
. 0183E  A5 B1     LDA $B1
. 01840  D0 1C     BNE $185E
. 01842  EA        NOP
. 01843  EA        NOP
. 01844  EA        NOP
. 01845  A2 00     LDX #$00
. 01847  A1 FA     LDA ($FA,X)
. 01849  20 75 18  JSR $1875
. 0184C  20 8F 18  JSR $188F
. 0184F  20 9B 18  JSR $189B
. 01852  C6 9C     DEC $9C
. 01854  D0 D1     BNE $1827
. 01856  A9 01     LDA #$01
. 01858  85 B1     STA $B1
. 0185A  4C 27 18  JMP $1827
. 0185D  EA        NOP
. 0185E  A9 00     LDA #$00
. 01860  85 B1     STA $B1
. 01862  A2 00     LDX #$00
. 01864  A1 FA     LDA ($FA,X)
. 01866  20 75 18  JSR $1875
. 01869  20 8F 18  JSR $188F
. 0186C  20 A9 18  JSR $18A9
. 0186F  C6 9B     DEC $9B
. 01871  D0 B0     BNE $1823
. 01873  00        BRK
. 01874  00        BRK
. 01875  A2 1F     LDX #$1F
. 01877  8E 00 D6  STX $D600
. 0187A  2C 00 D6  BIT $D600
. 0187D  10 FB     BPL $187A
. 0187F  8D 01 D6  STA $D601
. 01882  60        RTS
. 01883  18        CLC
. 01884  A5 FA     LDA $FA
. 01886  69 08     ADC #$08
. 01888  85 FA     STA $FA
. 0188A  90 02     BCC $188E
. 0188C  E6 FB     INC $FB
. 0188E  60        RTS
. 0188F  A0 28     LDY #$28
. 01891  A9 00     LDA #$00
. 01893  20 75 18  JSR $1875
. 01896  88        DEY
. 01897  D0 F8     BNE $1891
. 01899  60        RTS
. 0189A  EA        NOP
. 0189B  38        SEC
. 0189C  A5 FB     LDA $FB
. 0189E  E9 01     SBC #$01
. 018A0  85 FB     STA $FB
. 018A2  A5 FA     LDA $FA
. 018A4  E9 37     SBC #$37
```

```
. 018A6  85 FA     STA $FA
. 018A8  60        RTS
. 018A9  E6 FA     INC $FA
. 018AB  D0 02     BNE $18AF
. 018AD  E6 FB     INC $FB
. 018AF  60        RTS
. 018B0  EA        NOP
. 018B1  EA        NOP
. 018B2  EA        NOP
. 018B3  A9 00     LDA #$00
. 018B5  85 FC     STA $FC
. 018B7  A9 3F     LDA #$3F
. 018B9  85 FD     STA $FD
. 018BB  A2 19     LDX #$19
. 018BD  A9 80     LDA #$80
. 018BF  20 77 18  JSR $1877
. 018C2  EA        NOP
. 018C3  EA        NOP
. 018C4  A9 00     LDA #$00
. 018C6  A2 12     LDX #$12
. 018C8  20 77 18  JSR $1877
. 018CB  E8        INX
. 018CC  EA        NOP
. 018CD  EA        NOP
. 018CE  20 77 18  JSR $1877
. 018D1  A9 00     LDA #$00
. 018D3  20 75 18  JSR $1875
. 018D6  C6 FC     DEC $FC
. 018D8  D0 F7     BNE $18D1
. 018DA  C6 FD     DEC $FD
. 018DC  D0 F3     BNE $18D1
. 018DE  60        RTS
```

In questa spiegazione le istruzioni si riferiscono agli indirizzi di memoria che appaiono nella colonna a sinistra dell'output del monitor di linguaggio macchina, come per le spiegazioni dei programmi in linguaggio macchina. La prima istruzione (JSR $18B3) salta ad una subroutine per azzerare lo schermo bit map (8563). La sequenza successiva di istruzioni ($1803-$1809) pone il bit 7 del registro 25 dell'8563 per selezionare il modo bit map. Questo bit arriva automaticamente a 0 per la visualizzazione del testo. La subroutine che comincia in $1877 è la routine a voi familiare del registro di scrittura vista precedentemente in questo capitolo. Notate che se la subroutine fosse iniziata alla locazione $1875, avrebbe scritto nella RAM 8563 invece che in un registro dell'8563. A questa routine verrà richiesto di fare ciò più avanti nel programma. Il trucco per il successo di questo programma è indirizzare correttamente sia lo schermo VIC che lo schermo 8563. La seconda sequenza di istruzioni ($180A-$1816) definisce il byte alto dell'indirizzo a 16 bit per l'inizio di entrambi gli schermi VIC e 8563. Prima il registro X è caricato con il valore 18 ($12) che seleziona il registro 18 dell'8563, il puntatore alto dell'indirizzo della RAM 8563. L'accumulatore viene caricato con il valore $20, che è memorizzato nella locazione $FB, il puntatore bit map del VIC (byte alto). Ora l'accumulatore è portato a zero e la subroutine del registro di scrittura pone il valore zero nel puntatore bit map dell'8563 (byte alto). Le istruzioni dalla $1817 alla $181A compiono le stesse operazioni per il byte basso di entrambi i puntatori bit map del VIC e dell'8563. Le istruzioni tra $1820 e $1827 inizializzano le variabili $FC, $FD, $B1 e $9C a zero. Le variabili $FC e $FD sono i puntatori dei byte basso e alto degli indirizzi bit map della RAM 8563. La variabile $B1 è il contatore delle colonne dello schermo

VIC. La variabile $9C è il contatore che calcola quanti indirizzi saltare nella RAM 8563. Questo verrà spiegato più avanti nel programma. Ecco dove diviene importante una delle differenze maggiori tra i due chip video – il modo in cui il bit map corrisponde all'immagine sullo schermo. Come sapete, lo schermo VIC memorizza otto righe di 8 pixel ciascuna per formare una cella del carattere. Ogni riga degli otto pixel di un carattere dello schermo viene memorizzata consecutivamente. Per esempio, la posizione del carattere HOME è memorizzata nelle locazioni di default dalla $2000 alla $2007 per il bit map. La posizione del carattere successivo (come se il video fosse un testo standard) a destra della Posizione HOME è memorizzata tra $2008 e $2015. Così la prima riga di caratteri è memorizzata tra $2000 e $2A00 ($2000 più (8*8) * 40=2560). Ciò è rappresentato graficamente nella Figura 11-5. D'altra parte, l'8563 memorizza un bit map lungo le righe del raster sullo schermo. Ogni insieme di otto pixel di una riga del raster è memorizzata in una locazione della memoria della RAM 8563 che comincia a $0000. Per esempio, la prima riga del raster è memorizzata nelle locazioni dalla $0000 alla $0050, una locazione per ogni gruppo di otto pixel. Lo schermo dell'8563 è di 640 pixel in orizzontale; perciò ogni riga ha bisogno di 80 byte di memoria nella RAM 8563.

Ora vedete come i due bit map sono indirizzati in modo diverso. Per capire completamente come questo programma prenda i dati dal bit map del VIC e li ponga nel punto giusto del bit map dell'8563, viene fornito questo algoritmo. Le reali istruzioni per compiere questi passi vengono date più avanti. Ecco lo schema generale che dovete seguire:

1. Cominciate all'inizio delle locazioni di entrambi i bit map del VIC e dell'8563.
2. Prendete un byte del bit map del VIC.
3. Memorizzatelo nel bit map 8563.
4. Incrementate il puntatore del bit map VIC 8 volte (il puntatore dell'8563 si incrementa da solo dopo una scrittura).
5. Fate i passi 2, 3 e 4 39 volte. A questo punto, avete appena trasferito la prima riga del raster del bit map VIC sul bit map 8563.
6. Dopo che ogni riga del raster è trasferita, sottraete 311 dal puntatore del bit map VIC. Questo viene fatto 7 volte. L'ottava volta, il puntatore del bit map è incrementato di 1 per arrivare all'inizio della cella successiva del carattere VIC.
7. Eseguite 40 operazioni di scrittura di blank (scrivete zero nelle 40 colonne dei blank) per arrivare all'inizio di una riga raster dell'8563. Ricordate, il puntatore del bit map 8563 deve essere connesso al puntatore del bit map VIC.

Se il contatore di colonne del VIC è uguale a 39, incrementate il puntatore del bit map 8563 41 volte per porre la sua posizione relativa all'inizio di una riga di nuovi caratteri in relazione al bit map del VIC.

8. Ora ripetete il procedimento 25 volte, una per ogni riga di caratteri.

Nel programma, come abbiamo visto, le istruzioni dalla $180A alla $181A fanno iniziare i bit map VIC e 8563 rispettivamente alle locazioni $2000 e $0000. Tutte queste istruzioni comprendono il passo 1 dell'algoritmo.

Le istruzioni che compiono il passo 2 dell'algoritmo sono memorizzate nelle locazioni $182D–$1830. Il passo 3 memorizza il byte bit map del VIC nella posizione bit map dell'8563 ed è eseguito dall'istruzione $1831 e dalla subroutine memorizzate tra $1875 e $1882. Questa subroutine scrive il valore memorizzato nell'accumulatore nell'indirizzo appropriato della RAM 8563 come specificato dai contenuti dei registri 18 ($12) e 19 ($13) dell'8563. Le istruzioni memorizzate da $183A fino a $183D compiono il quarto passo dell'algoritmo. Quelle memorizzate da $1883 a $188E aggiornano il puntatore bit map del VIC a 8. I passi 2, 3 e 4 vengono compiuti mentre il contatore della riga di pixel in $9C è minore di sette. Se è minore di sette, il byte successivo del bit map del VIC è caricato e memorizzato nel bit map dell'8563. Il passo 6 è contenuto nella subroutine che inizia in $189B. Questa decrementa il puntatore bit map del VIC di 311 locazioni. Il passo 7 dell'algoritmo è contenuto nella subroutine memorizzata nelle locazioni da $188F a $1899. Infine, il passo 8 dell'algoritmo è compiuto dalle istruzioni da $186F a $1872.

Questo programma vi permette di trasferire un bit map in un altro, ma mantiene a 40 colonne il bit map del VIC. Espandete l'algoritmo in questa sezione e centrate il bit map sullo schermo ad 80 colonne. Scrivete una subroutine che espande in proporzione il bit map di 40 colonne e lo pone sull'intero schermo ad 80 colonne. Aggiungete anche una routine che azzera l'altra metà del bit map di 80 colonne. Aggiungete queste routine a qualsiasi applicazione gestionale, ed avrete uno strumento gestionale valido che potete usare completamente all'interno del chip 8563 che visualizza il vostro carattere ed i dati bit map.

SCROLLING DELLO SCHERMO 8563

L'8563 ha una funzione che permette allo schermo di muovere (scroll) dolcemente il suo contenuto sia in verticale (su o giù) che in orizzontale (sinistra o destra). La funzione di scrolling dell'8563 può essere considerata come una piccola finestra virtuale dello schermo che viene visualizzata su una parte dell'intero schermo di 80 per 25 caratteri. Lo schermo virtuale è visibile solo attraverso la finestra che voi definite nella specifica posizione dell'intero schermo. Entrambi gli schermi, tuttavia, esistono nelle locazioni di memoria della RAM 8563.

SCROLLING VERTICALE

Lo scrolling in direzione verticale è più facile dello scrolling in orizzontale. L'algoritmo generale per lo scrolling verticale è il seguente:

1. Ponete l'inizio del default della memoria del carattere della RAM a $2000 nella RAM 8563. Questo viene fatto scrivendo sui (o usando i contenuti del default dei) registri 12 ($0C) e 13 ($0D). Ora la locazione $2000 della RAM 8563 segna l'inizio del primo carattere nell'angolo in alto a sinistra della memoria video (la posizione HOME).

A questo punto, la misura dello schermo è 80 per 25 e le locazioni dello schermo RAM vanno da $2000 a $27CF nella RAM 8563. Perciò, i primi 80 byte memorizzano i primi 80 caratteri nella riga in alto dello schermo dell'8563.

2. Ora aggiungete 80 ($50) al byte basso del Puntatore video della RAM, registro 13. I secondi 80 byte della memoria video della RAM vengono visualizzati come primo carattere della riga. Lo schermo è ora definito dalle locazioni da $2050 a $281F nella memoria 8563. Questo muove i caratteri su di una riga di caratteri. La "vecchia" prima riga di caratteri si è mossa fuori dall'alto dello schermo e la "nuova" riga di caratteri si è mossa sull'ultima riga di caratteri dello schermo da una zona nell'area video che prima non era visibile. La nuova riga di caratteri dovrebbe essere definita nella RAM 8563 prima di essere mossa sullo schermo come ultima riga di caratteri.
3. Sebbene ciò muova un'intera riga di caratteri della RAM video su di una riga, il movimento è improvviso, inaspettato e non dolce. L'8563 rende lo scrolling verticale dolce, con i quattro bit più bassi del registro 24 ($18) dell'8563. Per fare lo scrolling dolce di una linea di scansione alla volta, scrivete il valore 1 nel registro 24. Questo muove l'intero schermo su di una linea di scansione. La linea di scansione che prima era visibile ora è fuori dallo schermo in alto ed una nuova linea di scansione è spostata da una locazione fuori dallo schermo in fondo sull'ultima linea di scansione visibile dello schermo.
4. Per lo scrolling di linee di scansione successive, incrementate il valore dei quattro bit più bassi del registro 24. Incrementando questo valore a 2, l'intero schermo viene spostato dolcemente di una ulteriore linea di scansione.
5. Continuate ad incrementare il valore del registro 24 finchè il valore è uguale a quello memorizzato nei quattro bit più bassi del registro 9, il numero totale delle linee di scansione verticali di un carattere. Una volta che questi valori sono uguali, solo la linea di scansione in fondo alla riga del primo carattere è visibile in alto nello schermo, e in fondo allo schermo è visibile tutto tranne l'ultima linea di scansione del dato da poco scrollato.
6. Per continuare lo scrolling, azzerate il valore dei quattro bit più bassi del registro 24 ed aggiungete 80 locazioni ai contenuti del puntatore video della RAM come avete fatto nel passo 2.
   Se volete usare gli attributi, dovete aumentare il valore del puntatore di inizio degli attributi anche per stare al passo con il puntatore video della RAM.

```
. 01800  A2 0C     LDX #$0C
. 01802  A9 20     LDA #$20
. 01804  85 FB     STA $FB
. 01806  20 60 18  JSR $1860
. 01809  E8        INX
. 0180A  A9 00     LDA #$00
. 0180C  85 FA     STA $FA
. 0180E  20 60 18  JSR $1860
. 01811  EA        NOP
. 01812  A2 18     LDX #$18
. 01814  A9 01     LDA #$01
. 01816  A8        TAY
. 01817  20 60 18  JSR $1860
. 0181A  EA        NOP
. 0181B  C8        INY
. 0181C  98        TYA
. 0181D  C0 08     CPY #$08
. 0181F  D0 F6     BNE $1817
. 01821  EA        NOP
. 01822  EA        NOP
```

```
. 01823  A9 00     LDA #$00
. 01825  20 60 18  JSR $1860
. 01828  A0 00     LDY #$00
. 0182A  E6 FA     INC $FA
. 0182C  D0 02     BNE $1830
. 0182E  E6 FB     INC $FB
. 01830  C8        INY
. 01831  C0 50     CPY #$50
. 01833  D0 F5     BNE $182A
. 01835  A2 0C     LDX #$0C
. 01837  A5 FB     LDA $FB
. 01839  20 60 18  JSR $1860
. 0183C  E8        INX
. 0183D  A5 FA     LDA $FA
. 0183F  20 60 18  JSR $1860
. 01842  A9 00     LDA #$00
. 01844  85 FD     STA $FD
. 01846  A9 00     LDA #$00
. 01848  85 FC     STA $FC
. 0184A  E6 FC     INC $FC
. 0184C  D0 FC     BNE $184A
. 0184E  E6 FD     INC $FD
. 01850  A5 FD     LDA $FD
. 01852  C9 A0     CMP #$A0
. 01854  D0 F0     BNE $1846
. 01856  4C 12 18  JMP $1812
. 01859  EA        NOP
. 0185A  EA        NOP
. 0185B  EA        NOP
. 0185C  EA        NOP
. 0185D  EA        NOP
. 0185E  EA        NOP
. 0185F  EA        NOP
. 01860  8E 00 D6  STX $D600
. 01863  2C 00 D6  BIT $D600
. 01866  10 FB     BPL $1863
. 01868  8D 01 D6  STA $D601
. 0186B  60        RTS
```

SCROLLING ORIZZONTALE

Lo scrolling orizzontale è più complesso dello scrolling verticale. Primo, la memoria Video della RAM deve essere azzerata per corrispondere alla misura "virtuale" dello schermo. Se lo schermo "virtuale" è largo 132 caratteri, allora la memoria Video della RAM deve essere organizzata come se si dovessero visualizzare 132 caratteri, anche se saranno visualizzati solo 80 caratteri orizzontali nella "window" dell'8563.

Questo significa che per visualizzare una window (finestra) di 25 righe di 80 caratteri di uno schermo virtuale di 132 caratteri di ampiezza, la memoria video della RAM deve avere almeno 3300 (132 per 25) locazioni di memoria e lo stesso numero di attributi.

Per visualizzare una window orizzontale di 80 caratteri, l'8563 deve prima leggere le prime 80 locazioni e attributi di memoria video dalla prima riga di 132 caratteri, e poi "saltare" le successive 52 (132-80) e procedere nel leggere i successivi 80 puntatori e attributi della seconda riga di 132 caratteri. La quantità del "salto" deve essere scritta in R27. In questo caso viene scritto "52" in R27 all'inizializzazione del sistema.

Quando lo scrolling orizzontale non viene usato, uno "zero" scritto in R27

porterà l'8563 a non saltare locazioni o attributi della memoria video, permettendo allo schermo "virtuale" nella memoria Video della RAM di avere la stessa larghezza della finestra dell'8563.

Ora che l'8563 è stato predisposto per visualizzare una finestra di 80 caratteri da uno schermo virtuale di 132 caratteri, può avvenire lo scrolling orizzontale modificando l'indirizzo in R12/R13 (INDIRIZZO DI INIZIO VIDEO) e R20/R21 (INDIRIZZO DI INIZIO ATTRIBUTI). Aumentando questi indirizzi di uno, ci sarà lo spostamento dell'intero schermo a "sinistra" di un carattere. Il primo carattere di ogni riga sarà spostato "fuori" dal lato sinistro dello schermo, tutti i caratteri dello schermo verranno mossi di uno spazio a sinistra, ed un carattere extra sarà spostato sul lato destro dello schermo. Una semplice manipolazione dell'indirizzo in R12/R13 e in R20/R21 sposterà la finestra a sinistra o a destra.

SCROLLING DOLCE ORIZZONTALE

L'8563 ha l'abilità di spostare dolcemente il testo in direzione orizzontale. Lo scrolling dolce orizzontale muove tutti i caratteri dello schermo del numero desiderato di pixel a "sinistra", fino alla larghezza di un carattere. Prima che possa avvenire lo scrolling dolce orizzontale, l'8563 deve avere qualcosa da spostare "sul" lato destro dello schermo, così R27 deve essere maggiore di zero. Questo predispone uno schermo virtuale maggiore di almeno un carattere rispetto allo schermo visualizzato.

Lo schermo è spostato di un pixel a sinistra portando i contenuti di R25 (3-0) a "uno" ed aumentando i contenuti di R22 (3-0) di uno. Entrambi i registri devono essere modificati per spostarsi orizzontalmente.

Il numero in R25 (3-0) e R22 (3-0) può essere aumentato, aumentando il numero dei pixel spostati, finchè R25 (3-0) è uguale a R22 (7-4). Questo è il massimo dello spostamento dolce orizzontale permesso. A questo punto esisterà solo un pixel (il destro) per il primo carattere di ogni riga di caratteri e l'ultimo carattere si sarà spostato sulla riga solo con un pixel (il destro) ancora non visibile. Per avere un ulteriore scrolling, il CPU deve portare l'R25 (3-0) a zero ed aumentare sia R12/R13 (INDIRIZZO DI INIZIO VIDEO) che R20/R21 (INDIRIZZO DI INIZIO ATTRIBUTI) di uno per lo scrolling orizzontale di un carattere. Questa sequenza avrà spostato dolcemente il testo a sinistra di un carattere.

DESCRIZIONE REGISTRO PER REGISTRO

Questa sezione descrive ogni registro dell'8563 in dettaglio per numero di registro. I bit significativi sono isolati e spiegati separatamente.

Le convenzioni usate nelle descrizioni dei registri sono le seguenti:

Rn (msb-lsb)

La R sta per "registro". La lettera n rappresenta il numero di registro, che va da 0 a 36. L'msb sta per "most significant bit" (bit più significativo) in quella particolare descrizione. Spesso si dà per scontato che il bit più significativo sia il bit 7; tuttavia, qui il bit 7 non è sempre il più significativo.

La notazione lsb sta per "least significant bit" (bit meno significativo) in una descrizione particolare di un registro. Generalmente, si dà per scontato che il bit meno significativo sia 0; ma qui il bit 0 non è sempre il meno significativo.

Se un singolo numero di bit è tra parentesi, allora questo è l'unico bit spiegato per quella funzione particolare.

Consultate questa sezione velocemente. Consultate il resto del capitolo per avere ulteriori conoscenze sulla programmazione del video ad 80 colonne.

R0 TOTALE ORIZZONTALE

È il numero dei caratteri, meno 1, tra impulsi successivi di sincronizzazione orizzontale. Include la parte visualizzata di una riga di caratteri, il margine orizzontale e l'intervallo di blanking durante gli impulsi di sincronizzazione orizzontale.

R1 ORIZZONTALE VISUALIZZATO

È il numero di caratteri visualizzati su ogni riga di caratteri. Questo numero pone la larghezza della parte di riga di caratteri visualizzata.

R2 POSIZIONE DI SINCRONIZZAZIONE ORIZZONTALE

È il numero di caratteri dall'inizio della parte visualizzata di una riga di caratteri all'inizio dell'impulso di sincronizzazione orizzontale. Questo registro controlla la posizione orizzontale del testo sullo schermo CRT. Questo numero dovrebbe essere maggiore del numero in R1 (orizzontale visualizzato).

R3 (3-0) AMPIEZZA ORIZZONTALE SINCRONIZZATA

È l'ampiezza dell'impulso di sincronizzazione orizzontale in caratteri, più 1.

R3 (7-4) AMPIEZZA VERTICALE SINCRONIZZATA

È l'ampiezza dell'impulso verticale sincronizzato in linee di scansione. Questo numero pone la durata dell'impulso di sincronizzazione verticale, che si può estendere dopo la fine della frame corrente nella frame seguente. Nel modo di sincronizzazione e video intercalati, dovrebbe essere programmato con un valore uguale al doppio del numero di linee di scansione desiderate.

R4 TOTALE VERTICALE

È il numero di righe di caratteri, meno 1, tra successivi impulsi di sincronizzazione verticale. Questo numero include le righe visualizzate, il margine verticale e gli intervalli di blanking durante gli impulsi di sincronizzazione verticale e determina l'andamento della sincronizzazione verticale.

R5 (4-0) REGOLAZIONE DEL TOTALE VERTICALE

È il numero di linee di scansione aggiunte alla fine della frame per una giusta regolazione dell'intervallo della sincronizzazione verticale. Nel modo di sincronizzazione e video intercalati, dovrebbe essere programmato con un valore uguale al doppio del numero delle linee di scansione desiderato.

R6 VERTICALE VISUALIZZATO

È il numero di righe di caratteri visualizzate in una frame. Questo numero pone l'altezza della parte visualizzata della frame.

R7 POSIZIONE DI SINCRONIZZAZIONE VERTICALE

È il numero di righe di caratteri, più 1, dalla prima riga di caratteri visualizzata all'inizio dell'impulso di sincronizzazione verticale. Questo numero dovrebbe essere maggiore del numero in R6 (verticale visualizzato).

R8 (1-0) CONTROLLO DEL MODO INTERCALAZIONE

Ci sono tre modi video di scansione del raster: non intercalato, sincronizzazione intercalata, e sincronizzazione e video intercalati. Il modo non intercalato è stato descritto prima. In questo modo, ogni linea di scansione è ricaricata nell'intervallo di sincronizzazione verticale. Questo modo è selezionato ponendo R8 (1-0) uguale a 00 o a 10.

Nel modo di sincronizzazione intercalata [R8 (1-0) = 01], i campi pari e dispari si alternano per generare le frame. La stessa informazione è visualizzata in entrambi i campi pari e dispari ma la misurazione del tempo di sincronizzazione verticale fa sì che le linee di scansione nei campi dispari siano scostati da quelli nei campi pari, di mezza linea di scansione. Questi spazi tra linee di scansione adiacenti vengono riempiti, risultando in un carattere di qualità maggiore sui monitor progettati per ricevere video intercalati. Nel modo sincronizzazione e video intercalati [R8 (1-0) = 11] i campi pari e dispari si alternano per generare le frame. I campi dispari visualizzano le linee di scansione dispari ed i campi pari visualizzano le linee di scansione pari. Ciò raddoppia la densità dei caratteri verticali dello schermo. Per visualizzare il doppio delle righe di caratteri, l'utente deve inizializzare nuovamente i registri dell'8563. Così l'ampiezza di sincronizzazione verticale [R3 (7-4)] e la regolazione del totale verticale [R5 (4-0)] dovrebbe essere programmata con valori pari. La durata di ognuno di questi eventi (in linee di scansione) sarà metà del valore programmato nei registri.

R9 (4-0) TOTALE DEL CARATTERE, VERTICALE

È il numero di linee di scansione, meno 1, in un carattere. Questo numero include la parte visualizzata di un carattere e lo spazio verticale tra caratteri sotto la parte visualizzata. Questo registro pone la dimensione verticale di un carattere e la dimensione verticale di una riga di caratteri.

R10 (4-0) CURSORE DELL'INIZIO DELLA LINEA DI SCANSIONE

È il numero della prima linea di scansione (in alto) del cursore di inversione video. Il numero 0 si riferisce alla prima linea di scansione (in alto) del carattere. Questo registro ed il registro 11 (sotto) definiscono l'altezza del cursore, e possono creare vari effetti, da un cursore blocco ad un cursore sottolineatura.

R10 (6-5) MODO CURSORE

Il cursore può essere programmato in quattro modi diversi programmando R10 (6-5) a 00 (cursore fisso), 01 (no cursore), 10 (cursore lampeggiante ad un sedicesimo dell'intervallo della frame), o 11 (cursore lampeggiante ad un trentaduesimo dell'intervallo della frame).

R11 (4-0) CURSORE DELLA FINE DELLA LINEA DI SCANSIONE

È il numero dell'ultima linea di scansione (in fondo) del cursore di inversione video, più 1.

R12, R13 INDIRIZZO DI INIZIO VIDEO (ALTO, BASSO)

L'indirizzo dei primi 8 pixel (a sinistra) della linea di scansione in alto della frame è definito da R12 e R13. Gli 8 bit più significativi sono in R12, i meno significativi in R13. Questi pongono l'indirizzo completo di 16 bit per i primi 8 pixel. Gli indirizzi degli insiemi successivi di 8 pixel su quella linea di scansione sono incrementati dall'indirizzo precedente. Il dato dei pixel bit map per una linea di scansione è preso dalle locazioni di memoria adiacenti. Una frame di 640 pixel orizzontali per 200 pixel verticali userà 16.000 byte di locazioni di memoria video della RAM 8563 da (R12, R13) a (R12, R13 + 15999).

Gli attributi bit map sono un insieme di byte della RAM Video, ognuno dei quali corrisponde ad una posizione del carattere sulla frame. L'attributo a 8 bit definisce ulteriori caratteristiche dell'area del carattere in quella posizione della frame. Queste caratteristiche (attributi) sono i seguenti: I bit 3-0 sono il fondo R, V, B e I rispettivamente. I bit 7-4 sono il fondo R, V, B e I rispettivamente. Per una frame di righe i di j caratteri ciascuna, i primi attributi j definiranno gli attributi per la prima riga (in alto) di caratteri, e gli attributi (i per j) dovranno essere definiti nella RAM video 8563. Per una frame bit map di 25 caratteri di altezza e di 80 caratteri di larghezza, avrete bisogno di 2000 attributi.

R12, R13 INDIRIZZO DI INIZIO VIDEO (ALTO, BASSO)

L'indirizzo per il puntatore del primo carattere (in alto a sinistra) della frame è definito in R12 e in R13. Gli otto bit più significativi sono in R12, i meno significativi sono in R13. Questi pongono l'indirizzo completo a 16 bit per quel primo puntatore. L'indirizzo dei puntatori dei caratteri successivi è incrementato dall'indirizzo precedente, così i caratteri adiacenti orizzontalmente hanno puntatori in locazioni di memoria adiacenti. Una frame di 25 righe di caratteri di 80 caratteri ognuna userà 2000 puntatori di locazioni di memoria video della RAM 8563 da (R12/13) a (R12/13 + 1999).

R14, R15 POSIZIONE DEL CURSORE

La posizione del cursore è posta da R14 e R15. Questi due registri contengono l'indirizzo a 16 bit del cursore. R14 è il bit più significativo, R15 il meno significativo. Se l'indirizzo di R14/R15 è l'indirizzo di un carattere all'interno della parte visualizzata della frame, allora quel carattere sarà visualizzato nel video inverso per quelle linee di scansione determinate da R10 (4-0) e R11 (4-0). Questi registri permettono all'8563 di essere programmato per un cursore "blocco" di video inverso, o per un cursore "sottolineatura". La locazione del cursore può essere cambiata modificando i contenuti dei registri R14 e R15.

R16, R17 PENNA OTTICA VERTICALE, ORIZZONTALE

L'8563 fornisce una funzione penna ottica. Quando le transizioni del pin di input LPEN passano dal basso all'alto, l'8563 trattiene i conteggi dei caratteri correnti verticali ed orizzontali nei due registri delle penne ottiche. Il conteggio verticale è trattenuto in R16. Le transizioni LPEN che avvengono sulla riga in alto della frame, tratterranno un 1 in R16. Le transizioni su righe seguenti tratterranno 2, 3, etc. Il conteggio orizzontale è trattenuto in R17. Le transizioni LPEN che avvengono sulla colonna più a sinistra dei caratteri tratterranno un 8. Le transizioni su colonne successive tratterranno 9, 10, etc. La condizione dei registri della penna ottica è trattenuta nel bit 6 del registro di stato. Se 0, i registri della penna ottica non sono stati trattenuti e contengono dati invalidi. Se 1, i registri contengono validi dati di trattenimento. Una lettura dal CPU di R16 o R17 ristabilisce il bit di registro di stato.

LETTURA E SCRITTURA SULLA MEMORIA VIDEO DELLA RAM 8563

R18, R19 LOCAZIONE DI AGGIORNAMENTO R31 DATI CPU

Il CPU comunica con la memoria RAM dell'8563 via registro indirizzo e registro dati dell'8563. Perchè il CPU legga una locazione di memoria, deve porre quell'indirizzo nei registri R18 (byte più significativo) e R19 (meno significativo).

L'8563 risponde eseguendo una lettura di quella locazione di memoria e ponendo il dato in R31, che il CPU può leggere in seguito. Durante il periodo in cui la "lettura" è in sospeso, ed il dato non è ancora valido in R31, il bit Update Ready del registro di stato (bit 7) sarà 0. Al completamento del ciclo di lettura, il dato in R31 sarà valido ed il bit sarà 1. Quando il CPU legge il dato in R31, l'8563 incrementa l'indirizzo in R18/R19 e compie una lettura di quell'indirizzo. Questo permette al CPU di leggere successive locazioni di memoria senza cambiare in continuazione gli indirizzi in R18/R19.

Perchè il CPU scriva il dato in una locazione di memoria, l'indirizzo deve essere scritto in R18/R19 come per una "lettura" descritta sopra. Seguendo la lettura automatica e dopo che il bit Update Ready è un 1, il CPU dovrebbe scrivere il dato desiderato in R31. L'8563 poi scrive il dato nella Memoria dell'8563 all'indirizzo definito da R18/R19, incrementa l'indirizzo in R18/R19, legge il dato dalla locazione di memoria incrementata e lo pone in R31. L'8563 poi pone il bit Update Ready a 1. Se il bit Update Ready è uno zero, tutte le volte l'accesso del CPU alla memoria dell'8563 è in sospeso. Dovranno essere evitati ulteriori accessi a R18, R19 o R31 finchè il bit Update Ready è un 1.

R20, R21 INDIRIZZO DI INIZIO ATTRIBUTI (ALTO, BASSO)

L'indirizzo per l'attributo del primo carattere (in alto a sinistra) della frame è definito in R20 e R21. Gli 8 bit più significativi sono in R20, i meno significativi in R21. Questi pongono l'indirizzo completo a 16 bit per quel primo attributo. Gli indirizzi degli attributi dei caratteri successivi sono incrementati dall'indirizzo precedente, così i caratteri adiacenti orizzontalmente hanno gli attributi in locazioni di memoria adiacenti. Una frame di 25 righe di caratteri di 80 caratteri ciascuna userà 2000 attributi nelle locazioni di memoria video della RAM 8563 da (R20/R21) a (R20/R21 + 1999).

R22 (3-0) CARATTERE VISUALIZZATO, ORIZZONTALE

Questo numero pone l'ampiezza della parte visualizzata del carattere, e definisce lo spazio orizzontale tra caratteri a destra della parte visualizzata.

Se il carattere definito non ha spazio orizzontale tra caratteri, allora R22 (3-0) dovrà essere inizializzato con un valore uguale al valore di (R22 (7-4) più1) che può rimanere costante anche con l'uso dello scrolling dolce orizzontale.

Se il carattere definito ha lo spazio orizzontale tra caratteri, allora R22 (3-0) dovrà essere inizializzato con un valore uguale al numero dei pixel (orizzontali) della parte visualizzata di un carattere, meno 1 e il valore di R22 (3-0) dovrà essere sostituito da R25 (3-0) per implementare lo scrolling dolce orizzontale.

Nel modo doppia larghezza del pixel, il valore scritto in R22 (3-0) dovrebbe essere 1 in più dei numeri descritti sopra.

R22 (7-4) TOTALE DEL CARATTERE, ORIZZONTALE

È il numero dei pixel (orizzontali) in un carattere, meno 1. Include la parte visualizzata di un carattere e lo spazio orizzontale tra caratteri a destra della parte

visualizzata. Questo registro pone la dimensione orizzontale di un carattere.

Nel modo doppia larghezza del pixel, il valore scritto in R22 (7-4) dovrebbe essere il numero dei pixel (orizzontali) in un carattere. Non sottraete 1 dal valore in questo modo.

R23 (4-0) CARATTERE VISUALIZZATO, VERTICALE

È il numero di linee di scansione, meno 1, della parte visualizzata di un carattere. Questo numero pone l'altezza della parte visualizzata del carattere, e definisce lo spazio verticale tra caratteri sotto la parte visualizzata. R23 (4-0) può essere uguale o minore di R9 (4-0). Se sono uguali, allora la parte visualizzata del carattere è uguale al carattere totale, e l'intero carattere viene visualizzato, senza spazio verticale tra i caratteri.

R24 (4-0) SCROLL DOLCE VERTICALE

L'improvviso scroll di un'intera riga di caratteri è un cambiamento troppo repentino per alcune applicazioni. L'8563 fornisce uno scrolling dolce verticale dello schermo una linea di scansione alla volta, fino ad un'intera riga di caratteri. Ciò determina lo scrolling verticale che permette allo schermo di essere spostato in su o giù dolcemente per quante righe di caratteri si vuole.

Per spostare lo schermo in "su" di una linea di scansione verticale, il CPU dovrebbe scrivere un 1 in R24 (4-0). Questo salterà la prima linea di scansione della prima riga di caratteri. Allora la prima riga di caratteri inizierà con la seconda linea di scansione, muovendola in "alto" di una linea di scansione. Tutte le righe di caratteri sulla frame saranno mosse in "su" di una linea di scansione. In fondo alla frame, sarà visualizzata una ulteriore linea di scansione di una "nuova" riga di caratteri. Questa visualizzerà realmente parti di "N+1" righe di caratteri invece dello standard N.

Il numero in R24 (4-0) può essere aumentato, aumentando il numero di linee di scansione spostate, finchè R24 (4-0) è uguale a R9 (4-0). Questo è il massimo dello scrolling dolce ammesso. A questo punto, esisterà solo una linea di scansione (in fondo) per la prima riga di caratteri e l'ultima riga di caratteri sarà stata spostata nella frame con solo l'ultima linea di scansione non visibile. Per un ulteriore scrolling, il CPU deve cambiare R24 (4-0) in 0, ed aumentare R12/R13 (indirizzo di inizio video) e R20/R21 (indirizzo di inizio attributo) per lo scrolling verticale di una riga di caratteri. Questa sequenza avrà spostato il testo dolcemente di una riga di caratteri.

R24 (5) RITMO DI LAMPEGGIO DEL CARATTERE

Quando gli attributi sono abilitati, il byte attributo per ogni carattere contiene un bit che controlla il lampeggio di quel carattere. Il ritmo col quale il carattere lampeggia è determinato da R24 (5). Se questo bit è uno 0, allora il ritmo di lampeggio del carattere è un sedicesimo del ritmo della frame, se è 1, è un trentaduesimo.

R24 (6) SCHERMO INVERSO

Lo schermo completo può essere invertito (sono invertiti i colori di primo piano e di fondo) modificando R24 (6). Con questo bit a 0, lo schermo sarà visualizzato con i colori normali di primo piano/fondo. Un 1 invertirà i colori di primo piano e di fondo per ogni carattere dello schermo. Questo cambio invertirà nuovamente ogni carattere normalmente invertito dal cursore e/o l'attributo di inversione per ogni carattere.

R24 (7) COPIA BLOCCO
R30 CONTATORE DELLE PAROLE

Per aumentare ulteriormente la velocità alla quale il CPU può manipolare la memoria dell'8563, sono state aggiunte due funzioni in più all'8563: Scrittura Blocco e Copia Blocco. La Scrittura Blocco è un'estensione del ciclo di scrittura del CPU tranne che la Scrittura Blocco scrive lo stesso dato in più di una locazione di memoria successiva. Questa operazione viene posta scrivendo l'indirizzo iniziale in R18/R19, aspettando che il bit di Stato Update Ready sia 1, scrivendo il dato in R31 ed aspettando ancora che il bit di Stato Update Ready sia 1. Questo è lo stesso della scrittura CPU (sopra).

Poi il CPU deve scrivere uno 0 in R24 (7) selezionando il modo Scrittura Blocco, poi scrive in R30 il numero di successive locazioni di memoria che l'8563 dovrebbe scrivere. Dopo la scrittura in R30, l'8563 inizializzerà uno o più cicli di scrittura nella memoria dell'8563, scrivendo i dati di R31 in successive locazioni di memoria. I contenuti di R18/R19 si incrementeranno quando saranno scritti gli indirizzi. Dopo che le Scritture Blocco sono completate, R18/R19 conterranno gli indirizzi che seguono le ultime locazioni di memoria scritte.

> **NOTA:** Un ciclo di scrittura seguirà la scrittura iniziale di dati in R31, così la quantità scritta in R30 dovrebbe essere uno in meno del numero totale di locazioni di memoria che si desiderano scrivere.

R25 (3-0) SCROLL DOLCE ORIZZONTALE

L'8563 può spostare il testo dolcemente in direzione orizzontale. Lo scrolling dolce orizzontale muove tutti i caratteri dello schermo a "sinistra" del numero desiderato di pixel, fino alla larghezza di un carattere. Prima che possa avvenire lo scrolling dolce orizzontale, l'8563 deve avere qualcosa da spostare "sulla" destra dello schermo, così R27 deve essere maggiore di 0. Ciò pone uno schermo virtuale di almeno un carattere più grande dello schermo visualizzato.

Uno schermo che non ha subito lo scrolling è ottenuto con il valore di R25 (3-0) uguale al valore di R22 (7-4) (totale del carattere, orizzontale). Lo schermo è spostato di un pixel a sinistra diminuendo i contenuti di R25 (3-0) di 1. Uno scrolling ulteriore avviene diminuendo i valori programmati. Si ottiene il massimo dello scrolling con un valore 0 in R25 (3-0). A questo punto, esisterà solo un pixel (il destro) per il primo carattere di ogni riga di caratteri e l'ultimo carattere si sarà spostato sulla riga con solo un pixel (il destro) non ancora visibile. Per un ulteriore scrolling, il CPU deve portare R25 (3-0) al suo valore massimo ed aumentare R12/R13 (Indirizzo di Inizio Video) e R20/R21 (Indirizzo di Inizio Attributi) ognuno di 1 per lo scrolling orizzontale di un carattere.

Questa sequenza sposterà dolcemente il testo di un carattere. Se il carattere ha qualche pixel non visualizzato {R22 (3-0) minore di [R22(7-4) più uno]} allora il valore di R22(3-0) (carattere visualizzato, orizzontale) deve essere modificato con R25(3-0) per lo scrolling orizzontale. Il valore di R22(3-0) dovrebbe essere uguale a [(il numero di pixel orizzontali nella parte visualizzata del carattere) più (il valore di R25(3-0), scroll dolce orizzontale)] modulo [(il valore di R22(7-4), totale del carattere, orizzontale) più 1]. In questo caso entrambi i registri R25(3-0) e R22(3-0) devono essere modificati per lo scrolling orizzontale. Per esempio, per un carattere di 8 pixel di larghezza con 5 pixel di larghezza visualizzati:

	R22(7-4)	**R25(3-0)**	**R22(3-0)**
Non scroll	**7**	**7**	**4**
Scroll 1	**7**	**6**	**3**
Scroll 2	**7**	**5**	**2**
Scroll 3	**7**	**4**	**1**
Scroll 4	**7**	**3**	**0**
Scroll 5	**7**	**2**	**7**
Scroll 6	**7**	**1**	**6**
Scroll 7	**7**	**0**	**5**

Ci sono due versioni diverse del chip 8563 (80 colonne RGBI), che richiedono valori di inizializzazione dei registri leggermente diversi.

Tra l'8563-R7A e le revisioni seguenti (8563-R8 e 8563-R9) c'è differenza nella funzione di scrolling orizzontale dolce, rappresentato dai bit 0-3 del registro 25. Anche se questa funzione non viene utilizzata, il valore corretto deve essere posto in questo registro per un video normale ad 80 colonne. Il software progettato specificamente per gli 8563-R7A mostrerà un problema nella parte più a sinistra del video quando elabora un sistema con R8 o R9, ed il software progettato specificamente per gli 8563R8 e R9 mostrerà un problema nella parte più a destra del video quando elabora un sistema con R7A. Per elaborare correttamente qualsiasi sistema, il software deve inizializzare il registro 25 con il dato corretto per la particolare versione del sistema dell'8563. La versione dell'8563 può venire facilmente accertata con il software leggendo il registro di stato dell'8563, localizzato in $D600 nella memoria I/O. I bit 0-2 contengono il numero della versione. Riferitevi alla tavola seguente per usare i dati reali.

8563rev	**versione($D600)**	**valore del registro 25**
7A	**0**	**$40**
8	**1**	**$47**
9	**1**	**$47**

Questo problema non riguarda il software che utilizza il sistema operativo interno al C128 (Kernal). Il Kernal determina esattamente la versione dell'8563 presente e l'inizializza di conseguenza. Il software che si basa sulla routine Kernal IOINIT per l'inizializzazione, non avrà problemi. Il software che compirà l'inizializzazione da solo, o che modificherà i contenuti del registro in causa, può mostrare le anomalie del video ad 80 colonne descritte, ma altrimenti funzionerà normalmente.

R25(5) PIXEL DI DOPPIA LARGHEZZA

Un cambiamento da una frame di 80 colonne ad una frame di 40 colonne richiede che i registri dell'8563 siano inizializzati in modo diverso e richiede un intervallo diverso di pixel. Per semplificare questo cambiamento, un bit di controllo nell'8563 divide il segnale di input DCLK per 2. Se R25(4) è uno 0, allora la larghezza del pixel è un periodo DCLK. Se è un 1, allora la larghezza del pixel è due periodi DCLK ed il periodo della clock interna è doppio del periodo normale. La temporizzazione AC è uguale a quella normale. I cicli del bus della RAM 8563 avvengono a metà dell'intervallo normale, con il tempo "non operativo" in atto con RAS e CAS "alti". Notate che i contenuti dei registri di inizializzazione orizzontale, ed i contenuti di R22(7-4) e R22(3-0) devono essere inizializzati con valori diversi in questo modo.

R25(5) MODO SEMIGRAFICO

La parte visualizzata di un carattere è limitata ad una larghezza di 8 pixel secondo l'ampiezza del Bus di Dati Video (DD0—DD7). Le grafiche a bassa risoluzione usano un set di caratteri "grafica di blocco" che possono superare gli 8 pixel di larghezza. L'8563 permette ciò in modo limitato, permettendo all'ultimo pixel visualizzato di un carattere di essere ripetuto nello spazio orizzontale tra caratteri, che normalmente è del colore di fondo. Se il bit di controllo R25(5) (modo semigrafico) è uno 0, avvengono normali operazioni dei caratteri. Se è un 1, avvengono operazioni semigrafiche, che permettono ai caratteri di estendersi nello spazio orizzontale tra caratteri, toccando il carattere successivo.

R25(6) ABILITAZIONE DEGLI ATTRIBUTI

In alcune applicazioni, gli attributi non sono necessari. In questi casi la quantità di memoria della RAM dell'8563 usata dagli attributi può essere eccessiva, così l'8563 ha un bit di controllo per disabilitare l'uso degli attributi. Se R25(6) è uno 0, allora gli attributi non verranno presi dall'8563 e la memoria della RAM può essere usata per altri scopi. In questo caso, nessun carattere può essere sottolineato, invertito individualmente o fatto lampeggiare. Può essere usato solo un insieme di 256 caratteri. Il colore di primo piano di tutti i caratteri sarà lo stesso e sarà determinato da R26(7-4). Il colore di fondo sarà posto ancora da R26(3-0). Se R25(6) è un 1, allora gli attributi saranno usati normalmente.

R25(7) SELEZIONARE IL MODO (Testo/Bit Map)

L'8563 è un dispositivo di visualizzazione di un testo, ma dispone anche di un modo grafico bit map limitato. Nel modo testo, la frame visualizzata è una matrice di caratteri, ognuna con un puntatore unico per riferirsi al carattere, ed un attributo per definire altre caratteristiche video. Nel modo bit map ogni pixel è controllato da un unico bit nella memoria della RAM dell'8563. Per un'operazione testo, il bit 7 è inizializzato a 0; per un'operazione bit map, ponete il bit 7. Si veda anche, in questo capitolo, la sezione bit-map del video ad 80 colonne.

R26(3-0) FONDO R, V, B, I

Il colore di fondo è determinato da R26(3-0). I bordi attorno allo schermo e gli spazi verticali ed orizzontali tra caratteri sono visualizzati nel colore di fondo.

R26(7-4) PRIMO PIANO R, V, B, I

Quando gli attributi sono disabilitati, il colore di primo piano per tutti i caratteri è determinato da R26(7-4).

R27 INCREMENTO DELL'INDIRIZZO PER RIGA

Un'ulteriore funzione dell'8563 è lo scrolling orizzontale, sia a sinistra che a destra nella memoria video della RAM 8563. Per spostare una finestra di i byte orizzontali per j linee di scansione verticali in uno schermo virtuale di k orizzontali per l verticali (i<k) ci devono essere k per l byte di dati bit map nella memoria video della RAM 8563. L'8563 deve "saltare" k−i byte su ogni linea di scansione, perchè solo i byte i di una linea viruale di byte k vengono visualizzati nella memoria video della RAM 8563. L'8563 può saltare un numero di byte, posti scrivendo in R27. Se R27 è 0, allora non saranno saltati byte. È necessario un valore che non sia zero in R27 per permettere lo scrolling dolce orizzontale.

Il valore in R27 viene usato per incrementare l'indirizzo dei dati bit map da una linea di scansione all'altra e per incrementare l'indirizzo degli attributi da una riga di caratteri all'altra. Sia i dati bit map che gli attributi saranno incrementati della stessa quantità, il valore di R27.

R28(4) TIPO DI RAM 8563(4416/4164)

Il dato per il testo visualizzato è memorizzato in una RAM dinamica destinata alla visualizzazione ed esterna all'8563. Il CPU accede a questa memoria video della RAM 8563 indirettamente attraverso i registri interni dell'8563. L'8563 calcola internamente 16 bit di indirizzi della RAM, così può indirizzare fino a 64K di byte della RAM 8563. Gli indirizzi della RAM 8563 sono compatibili con le RAM 4164 dell'8563 (64K per 1) o con le RAM 4416 dell'8563 (16K per 4). Nel registro 28, il bit 4 dovrebbe essere inizializzato a 0 per le operazioni con i 4416 e a 1 per le operazioni con i 4164.

4416:Riga/Colore = A7/A15, A6/A13, A5/A12, A4/A11, A3/A10, A2/A9,
A1/A8, A0/A8
4164:Riga/Colore = A7/A15, A6/A14, A5/A13, A4/A12, A3/A11,
A2/A10, A1/A9, A0/A8

R28(7–5) INDIRIZZO DI PARTENZA DI UN INSIEME DI CARATTERI

La locazione di partenza dell'insieme di caratteri nella RAM video 8563 è determinata da R28(7-5). Questi bit formano i 3 bit più significativi dell'indirizzo a 16 bit usato per accedere ai dati dei caratteri. Se R9(4-0) è maggiore di 15, allora saranno usati solo i bit 7-6 di R28, perchè ogni insieme di caratteri occuperà 8192 byte della memoria della RAM 8563.

I puntatori dei caratteri corrispondono ad una posizione del carattere visualizzata sulla frame. Il puntatore ad 8 bit seleziona uno dei 256 caratteri nell'insieme dei caratteri perchè venga visualizzato in quella posizione nella frame. Per una frame di i righe ognuna di j caratteri, i puntatori dei primi j definiranno i caratteri nella prima riga di caratteri (in alto), ed i puntatori (i per j) dovranno essere definiti nella RAM video dell'8563. Per 25 righe di caratteri ognuna di 80 caratteri, saranno necessari 2000 puntatori.

R29(4-0) CONTO DELLA LINEA DI SCANSIONE DI SOTTOLINEATURA

È il numero di linee di scansione usate per sottolineare. La sottolineatura è definita come una singola linea di scansione. La linea di scansione 0 si riferisce alla linea di scansione in cima al carattere.

R32, R33 INDIRIZZO DI PARTENZA DELLA COPIA BLOCCO

Copia Blocco è simile alla Scrittura Blocco, tranne che il dato scritto nella memoria dell'8563 è ottenuto da altre locazioni di memoria successive. La Copia Blocco "copia" realmente una parte della memoria dell'8563 in un'altra. Per iniziare questa operazione, il CPU scrive l'Indirizzo di Destinazione iniziale in R18/R19, aspetta che il bit di Stato Update Ready sia 1 e scrive un 1 in R24(7), selezionando il modo Copia Blocco. Poi il CPU scrive l'indirizzo di partenza in R32 (byte più significativo) e in R33 (meno significativo), poi il numero di successive locazioni di memoria da copiare dovrebbe essere scritto in R30. L'8563 leggerà allora i contenuti del primo indirizzo di partenza e lo scriverà nel primo indirizzo di destinazione. Si avranno ulteriori copie negli indirizzi incrementati dagli indirizzi di partenza e di destinazione, per tante parole quante sono definite in R30.

R34, R35 ABILITAZIONE VIDEO INIZIO, FINE

Durante gli impulsi di sincronizzazione orizzontale e verticale, i segnali R, G (V), B, I devono essere annullati (portati ad un livello basso) per impedire la visualizzazione del raggio di scansione durante i tempi di ritracciamento. Due registri permettono all'utente di regolare l'inizio (R34) e la fine (R35) dell'intervallo di blanking orizzontale.

Il registro R34 è programmato con il numero di caratteri dal primo carattere visualizzato di una riga al primo carattere annullato in quella riga. Il registro R35 è programmato con il numero di caratteri dal primo carattere visualizzato in una riga al primo carattere annullato in quella riga. Il blanking avviene in tutte le linee di scansione di una frame.

R36(3-0) RICARICA DELLA RAM DELL'8563/LINEA DI SCANSIONE

È il numero di cicli di ricarica della RAM Dinamica per tutte le linee di scansione. Questi cicli di ricarica avvengono su linee di scansione visualizzate e non visualizzate, sulle parti visualizzate dello schermo e le parti non visualizzate (linee di scansione annullate, confini verticali, e sincronizzazione verticale). Ogni indirizzo di ricarica è incrementato dal precedente, per tutti i 65.536 indirizzi. Il byte meno significativo dell'indirizzo della memoria Video a 16 bit della RAM 8563 è usato come indirizzo di riga nella RAM 8563. Quando l'indirizzo di ricarica a 16 bit è incrementato, vengono ricaricate righe successive della RAM 8563.

12

SUONO E MUSICA COL COMMODORE 128

INTRODUZIONE

Il Commodore 128 ha incorporato uno dei più sofisticati sintetizzatori del suono disponibili in un personal computer. Il sintetizzatore, chiamato il Sound Interface Device (SID), è un chip destinato solo a generare il suono e la musica. Il chip SID è in grado di produrre tre voci (suoni) indipendenti contemporaneamente. Ognuna delle voci può essere suonata in uno dei quattro tipi di suono, chiamati forme d'onda. Il chip SID ha anche i parametri Attacco (Attack), Caduta (Decay), Appoggio (Sustain), e Liberazione (Release) (ADSR) che sono programmabili. Questi parametri definiscono la qualità di un suono. Inoltre, il sintetizzatore ha un filtro che potete usare per selezionare certi suoni, eliminarne altri, o modificare le caratteristiche di un suono o di più suoni. Tutte queste funzioni equivalgono ad un sintetizzatore potente e versatile.

Per rendervi facile selezionare e manipolare le molte capacità del chip SID, il Commodore ha sviluppato istruzioni musicali in BASIC nuove ed efficaci.

Ecco le nuove istruzioni per il suono e per la musica disponibili sul C128:

SOUND
ENVELOPE
VOL
TEMPO
PLAY
FILTER

L'inizio di questa sezione spiega queste istruzioni per il suono, in formato enciclopedia. La seconda metà descrive come programmare il chip SID in linguaggio macchina.

ENVELOPE

Definisce un envelope degli strumenti musicali

ENVELOPE n,[,atk] [,dec] [,sus] [,rel] [,wf] [,pw]

dove: **n** numero di Envelope (0-9)
atk intervallo di Attack (Attacco) (0-15)
dec intervallo di Decay (Caduta) (0-15)
sus Sustain (Appoggio) (0-15)
rel intervallo di Release (Liberazione) (0-15)
wf Forma d'onda: 0=triangolo
1=dente di sega
2=impulso variabile (quadra)
3=rumore
4=modulazione del suono
pw Ampiezza dell'impulso (0-4095)

Un parametro non specificato manterrà il suo valore predefinito o ridefinito correntemente. L'ampiezza dell'impulso si riferisce solo all'ampiezza dell'impulso della forma d'onda variabile, o impulsiva, (wf=2). Il C128 ha inizializzati i seguenti 10 envelope:

	N A D S R	wf	pw	strumento
envelope	**0, 0, 9, 0, 0,**	**2,**	**1536**	**piano**
env.	**1,12, 0,12, 0,**	**1,**		**fisarmonica**
env.	**2, 0, 0,15, 0,**	**0,**		**organo a vapore**
env.	**3, 0, 5, 5, 0,**	**3,**		**tamburo**
env.	**4, 9, 4, 4, 0,**	**0,**		**flauto**
env.	**5, 0, 9, 2, 1,**	**1,**		**chitarra**
env.	**6, 0, 9, 0, 0,**	**2,**	**512**	**cembalo**
env.	**7, 0, 9, 9, 0,**	**2,**	**2048**	**organo**
env.	**8, 8, 9, 4, 1,**	**2,**	**512**	**tromba**
env.	**9, 0, 9, 0, 0,**	**0,**		**xilofono**

FILTER

Definisce i parametri del filtro del suono (chip SID)

FILTER [freq], [,lp] [,bp] [,hp] [,res]

dove: **freq** Frequenza di cutoff del filtro (0-2047)
dove: **lp** Filtro passa basso acceso (1), spento (0)
dove: **bp** Filtro passa banda acceso (1), spento (0)
dove: **hp** Filtro passa alto acceso (1), spento (0)
dove: **res** Risonanza (0-15)

I parametri non specificati non comportano cambiamenti dei valori correnti.

Potete usare più di un tipo di filtro per volta. Per esempio, sia il filtro passa basso (che lascia passare segnali a bassa frequenza) che quello passa alto (alta frequenza) possono essere usati insieme per avere una risposta filtrata (filtro a tacca). Perchè il filtro abbia un effetto udibile, deve essere selezionato almeno un tipo di filtro ed almeno una voce deve essere immessa attraverso il filtro.

ESEMPI:

FILTER 1024, 0, 1, 0, 2 Pone la frequenza di cutoff a 1024, seleziona il filtro passa banda e un valore di risonanza 2.

FILTER 2000, 1, 0, 1, 10 Pone la frequenza di cutoff a 2000, seleziona sia il filtro passa basso che il passa alto (per formare un filtro a tacca) e pone il valore di risonanza a 10.

PLAY

Definisce e suona gli elementi e le note musicali

PLAY "Vn, On, Tn, Un, Xn, elementi"

dove: **Vn** Voce (n=1-3)
On Ottava (n=0-6)
Tn Default dell'Envelope della Melodia (n=0-9)
0= piano
1= fisarmonica
2= organo a vapore
3= tamburo
4= flauto
5= chitarra
6= cembalo
7= organo
8= tromba
9= xilofono
Un Volume (n=0-15)
Xn Filtro acceso (n=1), spento (n=0)

Note: A, B, C, D, E, F, G (La, Si, Do, Re, Mi, Fa, Sol)

Elementi:
Diesis
$ Bemolle
W Semibreve
H Semitono
Q Semiminima
I Croma
S Semicroma
. punteggiato
R Pausa
M Attesa che tutte le voci che suonano al momento finiscano la battuta corrente.

L'istruzione PLAY vi dà la possibilità di selezionare la voce, l'ottava e l'envelope della melodia (inclusi 10 envelope degli strumenti musicali predefiniti), il volume e le note che volete suonare (PLAY). Tutti i controlli sono fra virgolette. Tutti gli elementi tranne R e M precedono le note musicali in una stringa PLAY.

ESEMPI:

PLAY "V1O4T0U5X0CDEFGAB" Suona le note C, D, E, F, G, A (Do, Re, Mi, Fa, Sol, La) e B (Si) nella voce 1, ottava 4, envelope di melodia 0 (piano), volume 5, senza filtro.

PLAY "V3O5T6U7X1#B$AW.CHDQEIF" Suona le note B (Si) diesis, A (La) bemolle, un C (Do) semibreve punteggiato, un C (Do) semitono, un E (Mi) semiminima, e F (Fa) croma.

SOUND

Crea effetti sonori e note musicali

SOUND v, f, d [,dir] [,m] [,s] [,w] [,p]

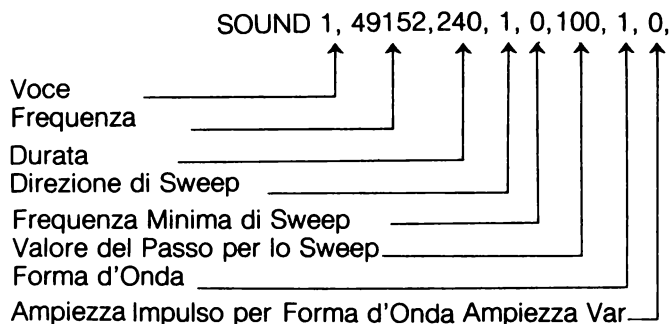
dove: **v** voce (1-3)
f valore di frequenza (0-65535)
d durata (0-32767)
dir default di direzione del passo (0 (su), 1 (giù) o 2 (oscillato))=0
m default di frequenza minima (se si usa uno sweep) (0-65535)=0
s default di valore del passo per lo sweep (0-32767)=0
w default della forma d'onda (0=triangolo, 1=dente di sega, 2=variabile (impulsuva), 3=rumore)=2
p default dell'ampiezza dell'impulso (0−4095)=2048

Il comando SOUND è un modo veloce e facile per creare effetti sonori e note musicali. I tre parametri richiesti v, f, d selezionano la voce, la frequenza, e la durata del suono. La durata è espressa in unità chiamate jiffies (momenti). Sessanta momenti equivalgono ad un secondo.

Il comando SOUND può fare lo sweep in una serie di frequenze che permettono agli effetti sonori di passare attraverso un intervallo di note. Specificate la direzione dello sweep con il parametro dir. Ponete il minimo della frequenza dello sweep ad m ed il valore del passo dello sweep ad s. Selezionate la forma d'onda appropriata con w e specificate con p l'ampiezza dell'impulso variabile della forma d'onda se selezionata in w.

ESEMPI:

SOUND 1, 40960, 60	Suona un SOUND alla frequenza 40960 con voce 1 per un secondo.
SOUND 2, 20000, 50, 0, 2000, 100	Emette un suono facendo lo sweep tra le frequenze cominciando a 2000 ed incrementandosi verso l'alto in unità di 100. Ogni frequenza è suonata per 50 momenti.
SOUND 3, 5000, 90, 2, 3000, 500, 1	Questo esempio emette una serie di suoni da una frequenza minima di 3000 fino a 5000, con incrementi di 500. Lo sweep è avanti e indietro (oscillante). La forma d'onda selezionata è dente di sega e la voce selezionata è 3.

ESEMPI:

```
10 DO
20 PRINT"VC  FREQ   DIR   MIN    VP  FD  AI ":PRINT
30 V=INT(RND(1)*3)+1    :REM VOCE
40 F=INT(RND(1)*65535) :REM FREQUENZA
50 D=INT(RND(1)*32767) :REM DURATA
60 DIR=INT(RND(1)*3)   :REM DIR CURVA
70 M=INT(RND(1)*65535) :REM FREQ MIN
80 S=INT(RND(1)*32767) :REM VAL PASSO
90 W=INT(RND(1)*4)     :REM FORMA D'ONDA
100 P=INT(RND(1)*4095) :REM AMP  IMPULSO
110 PRINT V; F;DIR; M;S;W;P:PRINT:PRINT
120 SOUND V, F, D, DIR, M, S, W, P
130 SLEEP 4
140 SOUND V, 0, 0, DIR, 0, 0, W, P
150 LOOP
```

TEMPO

Definisce la velocità del pezzo che viene suonato

TEMPO n

dove n è una durata relativa tra (0 e 255)
La reale durata di una semibreve è determinata usando la formula:

durata di una semibreve = 19.22/n secondi

Il valore di default è 8, e la durata della nota aumenta con n.

ESEMPI:

TEMPO 16	Definisce il TEMPO a 16
TEMPO 1	Definisce il TEMPO alla velocità minore
TEMPO 250	Definisce il TEMPO a 250

VOL

Definisce il livello di output del suono

VOL livello del volume

Questa istruzione pone il volume per le istruzioni SOUND e PLAY. Il livello VOLUME può essere posto da 0 a 15, dove 15 è il volume massimo, e 0 è spento. VOL incide su tutte le voci.

ESEMPI:

VOL 0 Pone il volume al suo livello più basso
VOL 15 Pone il volume per le frasi SOUND e PLAY all'output più alto.

CODIFICA DI UN BRANO DAL RIGO MUSICALE

Questa sezione fornisce un campione di musica scritta ed illustra come decodificare le note dal pentagramma e tradurle in una forma che il C128 possa capire. Ciò è sostanzialmente più veloce e più facile se sapete leggere la musica. Tuttavia, non dovete essere un musicista per essere in grado di suonare una melodia sul vostro C128. Per quelli tra voi che non sanno leggere la musica, la Figura 12-1 mostra come è fatto un pentagramma e in che relazione sono le note del pentagramma con i tasti di un pianoforte.

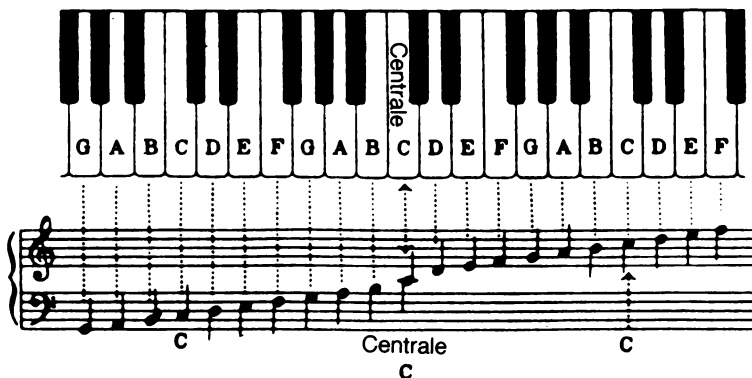



Figura 12-1. Pentagramma

La Figura 12-2 è un estratto da una composizione intitolata *Inventio 13* di Johann Sebastian Bach. Sebbene questa composizione sia stata scritta alcune centinaia di anni fa, può essere suonata con piacevole risultato sui più moderni sintetizzatori di computer, come il chip SID nel Commodore 128. Ecco le battute di apertura di *Inventio 13*.

Inventio 13

Figura 12-2. Parte di *Inventio 13* di Bach

Il modo migliore per cominciare a codificare una canzone sul vostro Commodore 128 è di analizzare le note in un codice intermedio. Scrivete le note del pentagramma superiore su un pezzo di carta. Ora scrivete le note del pentagramma inferiore. Fate precedere i valori delle note da un codice di durata. Per esempio, fate precedere una croma da un 8, una semicroma da un 16, e così via. Poi, suddividete le note in modo che le note del pentagramma superiore di una battuta siano proporzionali per durata con le note di una battuta del pentagramma inferiore.

Se la composizione musicale avesse un terzo pentagramma, suddividetelo in modo che la durata sia in proporzione con gli altri due pentagrammi. Una volta che le note di tutti i pentagrammi sono suddivise in durate uguali, una voce separata e determinata suona ogni nota di ciascun pentagramma. Per esempio, la voce 1 suona il pentagramma superiore, la voce 2 suona il secondo pentagramma e la voce 3 suona il pentagramma inferiore.

Per esempio se il pentagramma superiore comincia con una stringa di quattro crome, il pentagramma inferiore comincia con una stringa di otto semicrome, poichè una croma equivale per durata a due semicrome, separate le note come mostrato nella Figura 12-3.

V1 =	8A	8B	8C	8D
V2 =	16D 16E	16F 16G	16A 16B	16C 16D

Figura 12-3. Sincronizzazione di Note per Due Voci

La sincronizzazione e la temporizzazione in una composizione musicale sono critiche così dovete assicurarvi che le note del pentagramma superiore per la voce 1, siano, per esempio, in accordo di tempo con le note del pentagramma inferiore per la voce 2. La prima nota del pentagramma nella Figura 12-3 è un'A (La) croma. Le prime due note per la voce 2 sono D (Re) ed E (Mi) semicrome. In questo caso, dovete inserire per prima la voce 1 della croma nella stringa PLAY, poi fare seguire immediatamente la voce 2 delle semicrome. Per continuare l'esempio, la seconda nota nella Figura 12-3 per la voce 1 (pentagramma superiore) è un B (Si) croma. La croma Si è uguale per quanto riguarda il tempo a alle due semicrome F (Fa) e G (Sol) che appaiono nel pentagramma inferiore per la voce 2. Per coordinare il tempo, inserite la croma B (Si) nella stringa per la voce 1 e fatela seguire dalle due semicrome F (Fa) e G (Sol), per la voce 2.

Di regola, iniziate sempre con la nota di durata maggiore. Per esempio, se una battuta (misura) comincia con una serie di due semicrome nel pentagramma inferiore per la voce 2 ed il pentagramma superiore comincia con una croma per la voce 1, inserite la croma per prima nella stringa poichè deve suonare per il tempo in cui il Commodore 128 va a prendere le due semicrome. Dovete dare al computer il tempo di suonare per prima la nota più lunga, e poi di SUONARE (PLAY) le note più corte, altrimenti la composizione non sarà sincronizzata.

Ecco il programma che suona *Inventio 13*. Vengono omesse le spaziature nella stringa PLAY per risparmiare spazio. Per la leggibilità dei vostri programmi, aggiungete uno spazio tra gli elementi della stringa. Inseritela nel vostro C128 e salvatela (SAVE) per un uso futuro. Ora fatela girare (RUN), sedetevi e godetevi la musica!

```
10 REM INVENZIONE 13 DI BACH
20 TEMPO 6
30 PLAY"V104T7U8X0":REM VOCE 1=ORGANO
40 PLAY"V204T0U8X0":REM VOCE 2=PIANO
50 REM PRIME MISURE
60 A$="V201IAV103IEV202Q AV103SA04C03BEV202I#GV103SB04DV104ICV202SAEM"
70 B$="V104IEV202SA03CV103I#GV202SBEV104IEV202SB03D"
80 REM SECONDE MISURE
90 C$="V203ICV103SAEV202IAV103SA04CV202I#GV103SBEV202IEV103SB04D"
100 D$="V104ICV202SAEV103IAV202SA03CV104QRV202SBEB03D"
110 REM TERZE MISURE
120 E$="V203ICV104SREV202IAV104SCEV203ICV103SA04CV202IAV102SEG"
130 F$="V103IFV203SD02AV103IAV202SFAV104IDV202SDFV104IFV201SA02C"
140 REM QUARTE MISURE
150 G$="V201IBV104SFDV202IDV103SB04DV202IGV103SGBV202IBV103SDF"
160 H$="V103IEV202SGEV103IGV202SEGV104ICV202SCEV104IEV201SGB"
170 REM QUINTE MISURE
180 I$="V201IAV104SECV202ICV103SA04CV103IFV202SDFV104IDV201SB02D"
190 J$="V201IGV103SDBV201BV103SGBV103IEV202SCEV104ICV201SA02C"
200 REM SESTE MISURE
210 K$="V201IFV104SC03AV201IDV103SFAV103IDV201SG02GV103IBV202SFG"
220 M$="V201IAV104SC03AV202I#FV104SCEV201IBV104SD03BV202I#GV104SDF"
230 REM SETTIME MISURE
240 N$="V202ICV104SECV202IAV104SEGV202IDV104SFEV202I$BV104SDC"
250 O$="V202I#GV103SB04CV202IFV104SDEV202IDV104SFDV201IBV104S#GD"
260 REM OTTAVE MISURE
270 P$="V202I#GV104SBDV202IAV104SCAV202IDV104SFDV202IEV103SB04D"
280 Q$="V202IFV103S#GBV202I#DV104SC03AV202IEV103SEAV202IEV103SB#G"
290 REM NONE MISURE
300 R$="V20HAV103SAECE02QA"
310 PLAY A$:PLAY B$:PLAY C$:PLAY D$:PLAY E$
320 PLAY F$:PLAY G$:PLAY H$:PLAY I$:PLAY J$
330 PLAY K$:PLAY M$:PLAY N$:PLAY O$:PLAY P$
340 PLAY Q$:PLAY R$
```

Ecco due campioni di programmi di suoni che potete provare:

```
10 REM CICLI DI SUONO
20 REM VARIAZIONI DEL RITMO, DIREZIONE E FORMA D'ONDA
30 SV=1000:REM RITMO
40 DO
50 PRINT:PRINT "SV=";SV:PRINT
60 :    DIR=0 :REM DIREZIONE DELLA CURVA
70 :      DO
80 :        WF=0:REM FORMA D'ONDA
90 :           DO
100 :              SOUND 1,40000,120,DIR,2000,SV,WF
110 :           PRINT "WF=";WF;:WF=WF+1
120 :           LOOP UNTIL WF=4
130 :      PRINT "DIR=";DIR:DIR=DIR+1
140 :      LOOP UNTIL DIR = 3
150 SV=SV+2000
160 LOOP UNTIL SV > 20000
```

```
10 REM JOYSTICK MUSICALE - MODALITA' C128
20 REM ESEGUE LE NOTE DAL DO FINO AL SI IN CIASCUNA OTTAVA
30 REM CON L'INCREMENTO DEL VALORE DELLA NOTA PREMENDO IL JOYSTICK IN AVANTI
40 REM CON IL DECREMENTO DEL VALORE DELLA NOTA TIRANDOLO INDIETRO
50 REM CON L'INCREMENTO DI UN' OTTAVA SPINGENDO IL JOYSTICK A DESTRA
60 REM CON IL DECREMENTO DI UN' OTTAVA SPINGENDO IL JOYSTICK A SINISTRA
70 A$(1)="C":A$(2)="D":A$(3)="E":A$(4)="
80 Z=4:REM INIZIALIZZA LA QUARTA OTTAVA
90 SCNCLR :REM PULISCE LO SCHERMO
100 INPUT"INSERISCI LA VOCE 1";N$
110 INPUT"INSERISCI LA VOCE 2";O$
N20 INPUT"INSERISCI LA VOCE 3";P$
130 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT"                    INCREMENTO NOTA"
140 PRINT:PRINT"                         '":PRINT
150 PRINT"DECREM. OTTAVA  <-- -->  INCREM. OTTAVA"
160 PRINT:PRINT"                         V":PRINT
170 PRINT:PRINT"                DECREMENTO NOTA":PRINT
180 PRINT: PRINT"PER INCREMENTARE O DECREMENTARE UNA NOTA O UNA OTTAVA" ;
190 PRINT" SENZA MANDARE IN ESECUZIONE LENOTETENERE PREMUTOIL BOTTONEPER SPARARE"
200 PRINT" FINO A CHE NON SI SIA RAGGIUNTACONIL JOYSTICKL'APPROPRIATADIREZIONE"
220 Q$="V1"+"T"+N$:REM VOCE STRUMENTO 1
230 R$="V2"+"T"+O$:REM VOCE STRUMENTO 2
240 S$="V3"+"T"+P$:REM VOCE STRUMENTO 3
250 DO
260 : DO
270 :      A=JOY(1):REM ASSEGNAZIONE DEL VALORE DEL JOYSTICK
280 :      IF A=0 THEN LOOP:REM SE IL VALORE E' PARI A ZERO CONTINUA IL CICLO
290 :      B=A     :REM ASSEGNAZIONE DEL VALORE PER UNA SUCCESSIVA VERIFICA
300 :        Z$="O"+STR$(Z)
310 :        T$=Z$+Q$+A$(I):REM COSTRUZIONE DELLA VOCE 1
320 :        U$=Z$+R$+A$(I):REM COSTRUZIONE DELLA VOCE 2
330 :        V$=Z$+S$+A$(I):REM COSTRUZIONE DELLA VOCE 3
340 :
350 :        IF (A AND 128)=128 THEN 400
360 :          PLAY T$:REM ESECUZIONE VOCE 1
370 :          PLAY U$:REM ESECUZIONE VOCE 2
380 :          PLAY V$:REM ESECUZIONE VOCE 3
390 :

460 LOOP:REM ESECUZIONE DI UN ALTRO CICLO
```

SUONO E MUSICA IN MODO C128

BACKGROUND: LE CARATTERISTICHE DEL SUONO

Ogni suono che udite è in realtà un'onda sonora che si muove nell'aria. Come ogni onda, un'onda sonora può essere rappresentata graficamente e matematicamente (sinusoide) (Figura 12-4).

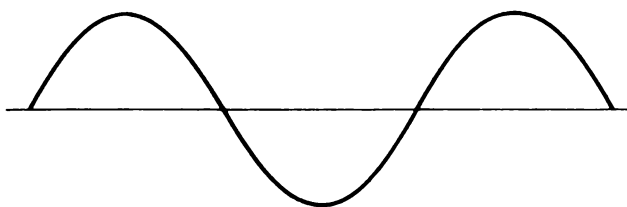



Figura 12-4. Sinusoide

L'onda sonora si muove (oscilla) con un particolare intervallo (frequenza) che determina il tono globale (altezza o bassezza del suono).

Il suono è costituito anche da armoniche che sono multiple della frequenza fondamentale del suono o della nota. La combinazione di queste onde sonore armoniche dà alla nota le sue qualità, chiamate timbro. La Figura 12-5 mostra la relazione tra frequenze sonore base ed armoniche.

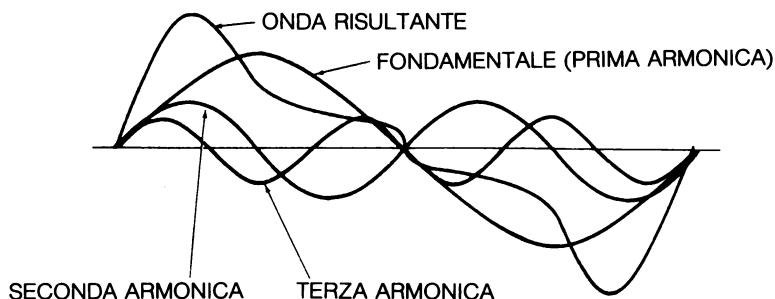



Figura 12-5. Frequenza ed Armoniche

Il timbro di un tono musicale, (cioè, il modo in cui un tono suona) è determinato dalla forma dell'onda del tono. Il C128 può generare quattro tipi di forme d'onda: triangolare, dente di sega, impulso variabile e rumore. La Figura 12-6 dà una rappresentazione grafica delle quattro forme d'onda.

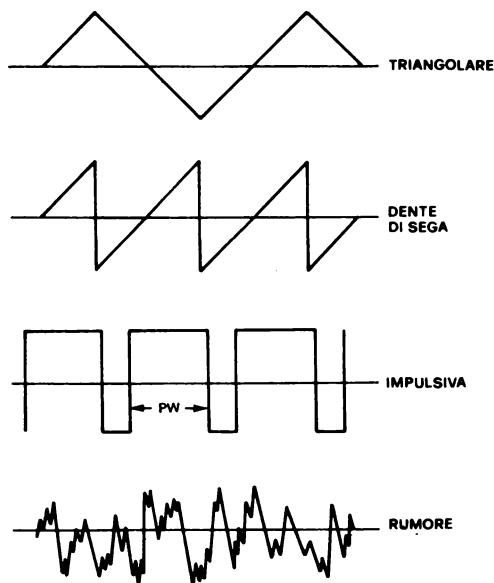



Figura 12-6. Tipi di Forma d'Onda Sonora

Il volume di un suono cambia per tutta la durata della nota, da quando la sentite per la prima volta finchè non è più udibile. Ci si riferisce a queste qualità del volume come Attacco (Attack), Caduta (Decay), Appoggio (Sustain), e Liberazione (Release) (ADSR). Attack è il ritmo con il quale una nota musicale raggiunge il massimo del suo volume. Decay è il ritmo col quale una nota musicale decresce dal massimo del suo volume verso il suo livello di metà estensione (sustain). Sustain è il livello al quale una nota musicale viene suonata a metà estensione del suo volume. Release è il ritmo col quale una nota musicale decresce dal suo livello sustain verso il volume zero. Il generatore ENVELOPE controlla i parametri del suono ADSR. Osservate la Figura 12-7 per una rappresentazione grafica di ADSR. Il Commodore 128 può cambiare ogni parametro ADSR in 16 diversi livelli. Questo dà un'assoluta flessibilità del generatore ENVELOPE e delle proprietà risultanti del volume del suono.

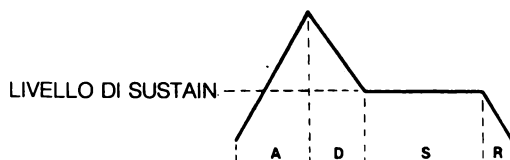



Figura 12-7. Fasi ADSR

Una delle istruzioni del suono più potenti del Commodore 128 – quella che controlla gli ADSR e la forma d'onda – è ENVELOPE. ENVELOPE pone i diversi comandi nel sintetizzatore che rendono particolare ogni suono. ENVELOPE vi dà il potere di manipolare il sintetizzatore SID.

Ecco le definizioni dei parametri all'interno della istruzione Envelope:

Envelope. Sono le proprietà di una nota musicale specificate dalla forma d'onda e dalle regolazioni attack, decay, sustain e release della nota. Per esempio, l'envelope per una nota di chitarra ha differenti forma d'onda e ADSR rispetto a quelli di una nota di flauto.

Waveform. È il tipo di onda sonora creata dalla combinazione di armoniche di accompagnamento di un tono. Le onde sonore armoniche di accompagnamento sono multiple della e sono basate sulla frequenza fondamentale del tono. Le qualità del tono generate da ogni forma d'onda sono nettamente diverse una dall'altra e sono rappresentate graficamente nella Figura 12-6.

Ampiezza dell'Impulso (pw). È la lunghezza del tempo tra le note della forma d'onda impulsiva.

Il Commodore 128 ha dieci envelope diversi predefiniti per dieci diversi strumenti musicali. Usando gli envelope predefiniti non dovete specificare i parametri ADSR, la forma d'onda e le regolazioni di durata dell'impulso: questo viene già fatto per voi. Tutto ciò che dovete fare è specificare il numero di envelope. Gli altri parametri vengono scelti automaticamente dal Commodore 128. Ecco gli envelope preselezionati per tipi diversi di strumenti musicali:

N. ENV.	STRUMENTO	A	D	S	R	FORMA D'ONDA	AMPIEZZA
0	Piano	0	9	0	0	2	1536
1	Fisarmonica	12	0	12	0	1	
2	Organo a vapore	0	0	15	0	0	
3	Tamburo	0	5	5	0	3	
4	Flauto	9	4	4	0	0	
5	Chitarra	0	9	2	1	1	
6	Cembalo	0	9	0	0	2	512
7	Organo	0	9	9	0	2	2048
8	Tromba	8	9	4	1	2	512
9	Xilofono	0	9	0	0	0	

Figura 12-8. Parametri di Default per l'Istruzione ENVELOPE

IL FILTRO SID

Una volta che avete selezionato ENVELOPE, ADSR, VOLume e TEMPO, usate il FILTER (FILTRO) per perfezionare i vostri suoni sintetizzati. Nel vostro programma, l'istruzione FILTRO (FILTER) deve precedere la frase PLAY. Prima dovete trovarvi a vostro agio nel generare il suono e preoccuparvi come ultima cosa dell'utilizzo del FILTER. Poichè il chip SID ha solo un filtro, esso si applica a tutte tre le voci. Le vostre melodie computerizzate verranno suonate senza il FILTRaggio, ma per sfruttare pienamente il sintetizzatore, usate FILTER per aumentare l'acutezza e la qualità del suono.

Nel paragrafo Le caratteristiche del suono, abbiamo definito un suono come un'onda che passa (oscilla) nell'aria con un particolare intervallo. L'intervallo nel quale un'onda sonora oscilla è chiamato frequenza d'onda. Ricordate che un'onda sonora è formata da una frequenza fondamentale e da armoniche di accompagnamento, che sono multiple della frequenza fondamentale. Osservate la Figura 12-5. Le onde armoniche di accompagnamento danno al suono il timbro, le qualità del suono che sono determinate dalla forma d'onda. Il filtro all'interno del chip SID vi dà l'abilità di accentuare ed eliminare le armoniche di una forma d'onda e di cambiarne il timbro.

Il chip SID filtra i suoni in tre modi: filtraggio passa basso, passa banda, e passa alto. Questi metodi di filtraggio sono aggiuntivi, ciò significa che potete usare più di un filtro per volta. Il passa basso taglia le frequenze sopra un certo valore da voi specificato, chiamato la frequenza di cutoff. La frequenza di cutoff è la linea di separazione che segna il limite del valore della frequenza che sarà suonata e di quella che non lo sarà. Nel filtraggio passa basso, il chip SID suona tutte le frequenze sotto la frequenza di cutoff e taglia le frequenze sopra essa. Come dice il nome stesso, le frequenze basse possono passare attraverso il filtro ma quelle alte no. Il filtro passa basso produce suoni pieni, omogenei. Osservate la Figura 12-9.

Figura 12-9. Filtro Passa Basso

Al contrario, il filtro passa alto permette a tutte le frequenze sopra la frequenza di cutoff di passare attraverso il chip. Tutte quelle sotto essa sono tagliate. Osservate la Figura 12-10. Il filtro passa alto produce suoni metallici, sordi.

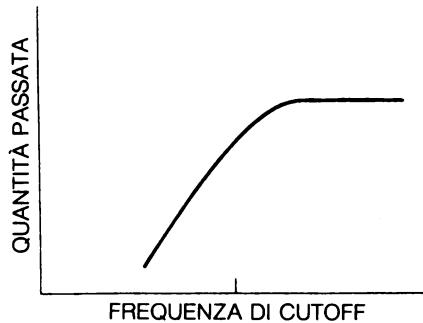



Figura 12-10. Filtro Passa Alto

Il filtro passa banda permette che una serie di frequenze passi attraverso il chip SID parzialmente sopra e sotto la frequenza di cutoff. Tutte le altre frequenze sopra e sotto la banda che circonda la frequenza di cutoff sono tagliate. Osservate la figura 12-11.

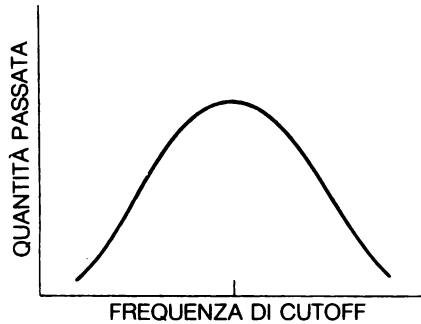



Figura 12-11. Filtro Passa Banda

FILTRAGGIO SUPERIORE

Ognuno degli esempi precedenti di filtraggio è solo un tipo di filtraggio per volta. Potete combinare i tre metodi di filtraggio del chip SID uno con l'altro per ottenere diversi effetti di filtraggio. Per esempio, potete abilitare i filtri passa basso e passa alto contemporaneamente per formare un filtro di a tacca. Un filtro a tacca permette alle frequenze sotto e sopra quella di cutoff di passare attraverso il chip SID, mentre le frequenze vicine alla frequenza di cutoff sono tagliate. Osservate la Figura 12-12 per una rappresentazione grafica di un filtro a tacca.

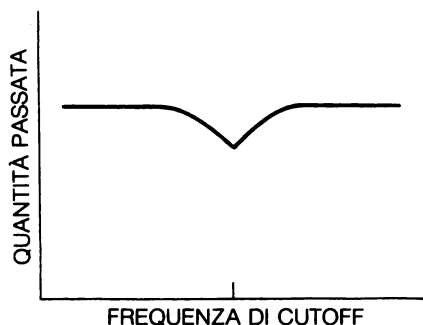



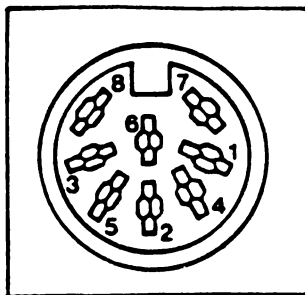
Figura 12-12. Filtro a Tacca

Potete anche aggiungere al filtro passa banda sia il filtro passa basso che quello passa alto per ottenere effetti interessanti. Unendo il filtro passa banda con il filtro passa basso, potete selezionare la banda di frequenze sotto la frequenza di cutoff e più sotto ancora. Tutte le altre sono tagliate. Unendo i filtri passa banda e passa alto, potete selezionare la banda di frequenze sopra la frequenza di cutoff e più in alto ancora. Tutte le frequenze sotto il cutoff sono tagliate.

Sperimentate le diverse combinazioni di filtri per vedere tutti i diversi tipi di accenti che potete porre sulle vostre note musicali e sugli effetti sonori. I filtri sono progettati per perfezionare i suoni creati dagli altri componenti del chip SID. Una volta che avete creato le note musicali o gli effetti sonori con il chip SID, tornate indietro ed aggiungete il filtraggio ai vostri programmi per renderli il più possibile nitidi e puliti.

Ora avete tutte le informazioni di cui avete bisogno per scrivere i vostri programmi musicali in BASIC sul Commodore 128. Sperimentate le diverse forme d'onda, le impostazioni ADSR, TEMPO e FILTER. Guardate in un libro di musica scritta su fogli ed inserite le note della scala musicale in sequenza all'interno di una stringa play. Accentate le note nella stringa con i caratteri di controllo SID. Potete unire il sintetizzatore del vostro Commodore 128 ai modi grafici del C128 per completare i vostri video o "film" con colonne sonore.

INPUT AUDIO NEL CHIP SID

Il chip SID ha un'ulteriore funzione poco conosciuta che la maggior parte dei computer non offre: Input audio. Questo vi permette di canalizzare la vostra musica, tipo quella della chitarra elettrica, e la vostra voce attraverso un microfono o qualsiasi altro strumento (ad alta impedenza, a livello di linea), che potete realmente "connettere" al C128. Per una corretta connessione al computer, dovete collegare il vostro cavo o comprarne uno che abbia gli attacchi giusti. Prendete un cavo con un connettore DIN a 8 piedini (pin) all'estremità che voi innestate nel C128. Connettete i pin 2 (terra), 3 (audio spento), 5 (audio acceso) all'estremità a 8 pin.
Ecco il pinout:

PIN	TIP	NOTAZIONE
1	LUM/SINC	Luminosità/SINC output
2	GND	
3	AUDIO SPENTO	
4	VIDEO SPENTO	Segnale di output composto
5	AUDIO ACCESO	
6	COLORE SPENTO	Segnale di output della cromaticità
7	NC	Non connesso
8	NC	Non connesso

Passate i tre fili metallici all'interno del cavo per connettervi all'audio acceso, audio spento e terra all'estremità del vostro connettore.

Potete connettere il C128 al vostro sistema stereo o VCR ed al suono canalizzato attraverso le casse acustiche. Fate attenzione, tuttavia, a non superare le limitazioni elettriche standard. Osservate le specificazioni sul chip SID nel Capitolo 17 sotto la descrizione EXT IN (Pin 26).

L'uso in questo modo del C128 vi fa capire come possa essere flessibile un personal computer e come possa comunicare con una grossa quantità di apparecchiature della vostra casa, incluso il telefono (con un ulteriore modem), il VCR, lo stereo, la TV, e non dimenticate la chitarra elettrica!

PROGRAMMAZIONE DEL CHIP SID IN LINGUAGGIO MACCHINA

Questa sezione fornisce un algoritmo e la procedura codificata del programma per suonare sul C128.

Prima dovete conoscere i tasti delle locazioni SID nella mappa della memoria del C128. In figura 12-13 diamo una mappa semplificata della memoria del SID che è esattamente la stessa sia in modo C128 che in modo C64. Il registro 0 è localizzato all'indirizzo 54272 ($D400).

Figura 12-13. Mappa della Memoria del SID

	INDIRIZZO A_4	A_3	A_2	A_1	A_0	REG # (HEX)	DATO D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	NOME DEL REGISTRO	TIPO DI REGISTRO
															Voce 1	
0	0	0	0	0	0	00	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ BASSA	SOLO SCRITTURA
1	0	0	0	0	1	01	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ ALTA	SOLO SCRITTURA
2	0	0	0	1	0	02	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW BASSO	SOLO SCRITTURA
3	0	0	0	1	1	03	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW ALTO	SOLO SCRITTURA
4	0	0	1	0	0	04	NOISE	⊓⊔⊓	/|/|	/\/\	TEST	RING MOD	SYNC	GATE	REGISTRO CONTROLLO	SOLO SCRITTURA
5	0	0	1	0	1	05	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	SOLO SCRITTURA
6	0	0	1	1	0	06	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	SOLO SCRITTURA
															Voce 2	
7	0	0	1	1	1	07	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ BASSA	SOLO SCRITTURA
8	0	1	0	0	0	08	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ ALTA	SOLO SCRITTURA
9	0	1	0	0	1	09	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW BASSO	SOLO SCRITTURA
10	0	1	0	1	0	0A	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW ALTO	SOLO SCRITTURA
11	0	1	0	1	1	0B	NOISE	⊓⊔⊓	/|/|	/\/\	TEST	RING MOD	SYNC	GATE	REGISTRO CONTROLLO	SOLO SCRITTURA
12	0	1	1	0	0	0C	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	SOLO SCRITTURA
13	0	1	1	0	1	0D	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	SOLO SCRITTURA
															Voce 3	
14	0	1	1	1	0	0E	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ BASSA	SOLO SCRITTURA
15	0	1	1	1	1	0F	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ ALTA	SOLO SCRITTURA
16	1	0	0	0	0	10	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW BASSO	SOLO SCRITTURA
17	1	0	0	0	1	11	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW ALTO	SOLO SCRITTURA
18	1	0	0	1	0	12	NOISE	⊓⊔⊓	/|/|	/\/\	TEST	RING MOD	SYNC	GATE	REGISTRO CONTROLLO	SOLO SCRITTURA
19	1	0	0	1	1	13	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	SOLO SCRITTURA
20	1	0	1	0	0	14	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	SOLO SCRITTURA
															Filtro	
21	1	0	1	0	1	15	—	—	—	—	—	FC_2	FC_1	FC_0	FC BASSO	SOLO SCRITTURA
22	1	0	1	1	0	16	FC_{10}	FC_9	FC_8	FC_7	FC_6	FC_5	FC_4	FC_3	FC ALTO	SOLO SCRITTURA
23	1	0	1	1	1	17	RES_3	RES_2	RES_1	RES_0	FILTEX	FILT 3	FILT 2	FILT 1	RES/FILT	SOLO SCRITTURA
24	1	1	0	0	0	18	3 OFF	HP	BP	LP	VOL_3	VOL_2	VOL_1	VOL_0	MODO/VOL	SOLO SCRITTURA
															Misc.	
25	1	1	0	0	1	19	PX_7	PX_6	PX_5	PX_4	PX_3	PX_2	PX_1	PX_0	POT X	SOLO LETTURA
26	1	1	0	1	0	1A	PY_7	PY_6	PY_5	PY_4	PY_3	PY_2	PY_1	PY_0	POT Y	SOLO LETTURA
27	1	1	0	1	1	1B	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0	OSC3/RANDOM	SOLO LETTURA
28	1	1	1	0	0	1C	E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	ENV3	SOLO LETTURA

Ecco l'algoritmo generale per emettere il suono dal chip SID in voce 1.

1. Azzerate il chip del suono (Per $D400-$D41C)
2. Selezionate ATTACK e DECAY per ogni voce ($D405)v1
3. Selezionate SUSTAIN e RELEASE per ogni voce ($D406)v1
4. Selezionate una forma d'onda ($D404 per v1)
5. Ponete il volume ($D418)
6. Ponete la frequenza della nota con un byte basso ($D400) ed un byte alto ($D401) dalla Tabella delle Note nella Figura 12-15 a fine capitolo.

Se volete emettere un certo valore di frequenza, usate questa equazione:

F_n = F out/.06097

Arrotondate per eccesso la frequenza (F_n) con la funzione intera e dividete il risultato nei byte basso ed alto.
In decimali:

F_{HI} = F_h Fn/256 è il byte alto (F_{HI} = alta frequenza)

e:

F_{low}= Fn −(256* F_{HI}) è il byte basso (F_{low} = bassa frequenza)

Per familiarizzarsi con la gamma delle note, provate con il comando SOUND in BASIC.

7. Ponete il byte alto nel Registro di Controllo della Frequenza (in $D401) per la voce appropriata.
8. Ponete il byte basso nel Registro di Controllo della Frequenza (byte basso) per la voce appropriata ($D400).
9. Inizializzate (GATE) la nota od il tono. Gating significa iniziare i cicli ATTACK, DECAY e SUSTAIN del tono. Ponete il bit 0 di $D404 (V1), $D40B (V2) e $D412 (V3), rispettivamente.
10. Lasciate inserito il bit 0 per la durata per cui volete suoni la nota od il tono.
11. Azzerate il bit GATE.

La durata delle note musicali standard è la seguente:

TIPO DI NOTA	DURATA
1/16	128
1/8	256
PUNTEGGIATA 1/8	384
1/4	512
1/4 + 1/16	640
PUNTEGGIATA 1/4	768
1/2	1024
1/2 + 1/16	1152
1/2 + 1/8	1280
PUNTEGGIATA 1/2	1536
SEMIBREVE	2048

Per sincronizzare tre voci, dividete ciascuna battuta in 16 parti uguali. Eseguite prima il Gate delle note più lunghe, poi il Gate dei valori più brevi, ma di uguale proporzione. Per esempio, un pentagramma ha una croma. Il pentagramma sotto ad esso ha due semicrome. Suonate la nota che dura di più così può subire il gate mentre le altre note più brevi sono recuperate.

Il seguente programma in linguaggio macchina suona le note ad una voce. Il dato della nota musicale comincia alla locazione $1890. Il dato di ogni nota è memorizzato nel formato a quattro byte che comincia in $1890 come segue:

Frequenza (byte basso/byte alto)

Durata (byte basso/byte alto)

Per esempio, il dato della seconda nota comincia alla locazione $1894, il dato della terza nota comincia alla locazione $1898 e così via.

Per ora sono stati forniti dei dati campione. Infine, ponete l'inizio dei vostri dati alla locazione $1890. Osservate la figura 12-15 alla fine di questo capitolo per i valori di frequenza del byte basso e del byte alto che devono essere posti nei registri di controllo della frequenza del SID. Fate riferimento alla tabella della durata appena vista per i valori dei byte della durata delle note.

Le istruzioni NOP sono aggiunte tra le sequenze di istruzioni per la leggibilità.

Ecco il listato del programma come appare nel Monitor di Linguaggio Macchina:

```
READY.

MONITOR
    PC  SR AC XR YR SP
; FB000 00 00 00 00 F8

. 01800  A2 00     LDX #$00
. 01802  8A        TXA
. 01803  9D 00 D4  STA $D400,X
. 01806  E8        INX
. 01807  E0 19     CPX #$19
. 01809  D0 F8     BNE $1803
. 0180B  EA        NOP
. 0180C  EA        NOP
. 0180D  EA        NOP
. 0180E  A2 05     LDX #$05
. 01810  A9 F5     LDA #$F5
. 01812  9D 00 D4  STA $D400,X
. 01815  EA        NOP
. 01816  EA        NOP
. 01817  EA        NOP
. 01818  A2 06     LDX #$06
. 0181A  A9 A3     LDA #$A3
. 0181C  9D 00 D4  STA $D400,X
. 0181F  EA        NOP
. 01820  EA        NOP
. 01821  EA        NOP
. 01822  EA        NOP
. 01823  A2 18     LDX #$18
. 01825  A9 0F     LDA #$0F
. 01827  9D 00 D4  STA $D400,X
. 0182A  EA        NOP
. 0182B  EA        NOP
. 0182C  EA        NOP
. 0182D  A0 00     LDY #$00
. 0182F  A2 00     LDX #$00
. 01831  B9 90 18  LDA $1890,Y
```

```
. 01834   F0 3F      BEQ $1875
. 01836   9D 00 D4   STA $D400,X
. 01839   E8         INX
. 0183A   C8         INY
. 0183B   B9 90 18   LDA $1890,Y
. 0183E   9D 00 D4   STA $D400,X
. 01841   EA         NOP
. 01842   EA         NOP
. 01843   EA         NOP
. 01844   C8         INY
. 01845   B9 90 18   LDA $1890,Y
. 01848   85 FA      STA $FA
. 0184A   C8         INY
. 0184B   B9 90 18   LDA $1890,Y
. 0184E   85 FB      STA $FB
. 01850   A2 04      LDX #$04
. 01852   A9 21      LDA #$21
. 01854   9D 00 D4   STA $D400,X
. 01857   EA         NOP
. 01858   EA         NOP
. 01859   EA         NOP
. 0185A   EA         NOP
. 0185B   C6 FA      DEC $FA
. 0185D   D0 FC      BNE $185B
. 0185F   C6 FB      DEC $FB
. 01861   D0 F8      BNE $185B
. 01863   EA         NOP
. 01864   A2 04      LDX #$04
. 01866   A9 20      LDA #$20
. 01868   9D 00 D4   STA $D400,X
. 0186B   EA         NOP
. 0186C   EA         NOP
. 0186D   EA         NOP
. 0186E   C8         INY
. 0186F   4C 2F 18   JMP $182F
. 01872   EA         NOP
. 01873   EA         NOP
. 01874   EA         NOP
. 01875   A2 18      LDX #$18
. 01877   A9 00      LDA #$00
. 01879   9D 00 D4   STA $D400,X
. 0187C   00         BRK


>01890 1E 19 80 00 1C 41 80 00
>01898 1E 19 80 00 1C 41 80 00
>018A0 1E 19 80 00 1C 41 00 00
>018A8 1E 19 80 00 1C 41 80 00
>018B0 1E 19 80 00 1E 19 80 00
>018B8 1E 19 80 01 1E 19 80 01
>018C0 1E 19 80 04 1C 19 80 04
>018C8 1E 19 80 04 1C 41 80 00
>018D0 1E 19 80 00 1C 41 00 00
>018D8 21 21 00 00 00 00 00 00
>018E0 0D 54 08 D0 D7 AD 7B 08
>018E8 8D 6C 08 AD 7C 08 8D 6D
```

Inserite il programma in memoria con il Monitor di Linguaggio Macchina e salvatelo. Fate girare il programma con il seguente comando:

G F1800

Il programma suona una serie di note con una singola voce (voce 1). Ecco una descrizione riga per riga delle istruzioni nel programma.

Le istruzioni memorizzate nelle locazioni dalla $1800 alla $180A azzerano i registri SID ($D400-$D418). Si consiglia questa pratica di programmazione per far sì che tutti i registri SID siano inizializzati a zero.

Le istruzioni memorizzate nelle locazioni dalla $180E alla $1814 assegnano i valori attack/decay al generatore envelope del SID. In questo programma, attack è posto a (15:$0F) e decay è posto a 5. Questo caricamento attack dà l'impressione che i suoni siano lontani e che stiano avvicinandosi, come il suono di un treno che si avvicina da lontano.

Le istruzioni memorizzate nelle locazioni dalla $1818 alla $181E selezionano i valori sustain/release per il generatore envelope della voce 1. In questo caso la durata di sustain è posta a 10 ($0A) e release è posto a 3. Per ulteriori informazioni sulle impostazioni ADSR, andate all'inizio del capitolo.

La sequenza successiva di istruzioni ($1823-$1829) pone il volume al massimo della potenza (15:$0F). Notate che le ultime tre sequenze di istruzioni hanno caricato il registro X con il numero di registro, hanno caricato l'accumulatore con il valore appropriato ed hanno memorizzato il valore nella locazione dell'inizio del SID più un offset nel registro X. Questo tipo di programmazione è facile da seguire ed è standardizzato per tutto il programma.

Le istruzioni memorizzate nelle locazioni dalla $182D alla $183E prendono i valori delle frequenze del byte basso ed alto per il dato della nota musicale, e li pongono nei registri di controllo della frequenza dei byte basso ed alto della voce 1. Prima i registri X e Y vengono azzerati. Il registro Y è usato come indice per il dato della nota musicale in memoria che comincia alla locazione $1890. Per accedere a byte successivi per la frequenza e la durata, il registro X viene incrementato. Il registro X viene usato come indice per accedere ai registri di controllo della frequenza dei byte basso ed alto alle locazioni $D400 e $D401. In questo segmento di programma, il registro X può avere solo uno dei due valori; 0 per accedere alla locazione $D400 e 1 per accedere alla locazione $D401.

Durante il ciclo, la prima volta l'istruzione che inizia alla locazione $1831 (LDA $1890, Y) carica il valore del dato musicale (frequenza del byte basso) nell'accumulatore. L'istruzione seguente (BEQ $1875) controlla che questo valore sia zero. Se lo è, il controllo passa alla locazione $1875, il volume è azzerato ed il programma finisce. Questo suggerisce che per terminare il programma voi poniate uno zero nel dato del byte basso. Questo meccanismo funziona come fine del dato, poichè non vengono letti altri dati se il valore del byte basso di una nota musicale è zero. Se il valore del dato non è zero, esso è memorizzato nel registro di controllo della frequenza del byte basso per la voce 1.

Sia il registro X che Y sono incrementati, ed il valore del dato musicale successivo, questa volta in $1891, è caricato nell'accumulatore con l'istruzione che inizia alla locazione $183B. L'istruzione successiva (STA $D400, X) memorizza il valore del dato nel registro di controllo della frequenza del byte alto ($D401) per la voce 1.

L'istruzione alla locazione $1844 incrementa il registro Y. L'istruzione successiva di caricamento che inizia alla locazione $1845 carica l'accumulatore con il dato successivo in memoria, questa volta è il byte basso per la durata della nota. Trovate la tabella della durata delle note a pag. 93. La durata del byte basso è memorizzata alla pagina zero della locazione $FA. Il registro Y è incrementato di nuovo e la durata del byte alto è caricata nell'accumulatore e memorizzata alla pagina zero della locazione $FB. Queste due locazioni sono decrementate nelle istruzioni memorizzate nelle locazioni dalla $185B alla $1862.

Le istruzioni memorizzate nelle locazioni dalla $1850 alla $1857 fanno il GATE della nota, in altre parole la suonano. Per emettere note udibili, ogni voce ha un bit gate che inizializza il suono del generatore envelope per una voce particolare. Il bit 0 della locazione $D404 è il bit gate per la voce 1. Per fare il gate di un bit, ponete contemporaneamente il bit 0 ed il bit della forma d'onda desiderata, che, in questo caso, è la dente di sega in (32+1 = $21).

A questo punto, viene fatto il gate della forma d'onda dente di sega che suona nel generatore envelope della voce 1. La frequenza selezionata viene emessa finchè spegnete il bit gate. Adesso entra in gioco la durata della nota. Le istruzioni memorizzate nelle locazioni dalla $185B alla $1862 decrementano il valore della durata del byte basso ed alto memorizzati alla pagina zero nelle locazioni $FA e $FB rispettivamente. Queste istruzioni funzionano come ciclo di ritardo nei cicli utilizzando i valori di durata della pagina 93 che sono memorizzati nella zona dei dati musicali che iniziano in $1890 della memoria. Quando sia $FA che $FB sono uguali a zero, il controllo del programma cade fino alle istruzioni memorizzate da $1864 a $186A. Queste istruzioni azzerano il bit gate della voce 1 e impediscono al generatore envelope di emettere il suono.

A questo punto, il registro Y è incrementato ($186E) ed il programma salta a $182F dove il registro X viene azzerato. Poichè il registro Y è già stato incrementato prima dell'istruzione JMP nella locazione $186F-71, il valore della frequenza del byte basso per la seconda nota è già puntato dal registro Y più l'indirizzo di base di $1890. Poi il programma carica quel valore del byte basso e lo memorizza alla locazione $D400 più l'offset del registro X, che in questo caso è 0. Ciò dà l'indirizzo appropriato $D400 al registro di controllo della frequenza (byte basso) della voce 1. Queste istruzioni vengono eseguite ripetutamente finchè non viene incontrata una frequenza zero del byte basso. Quando la frequenza del byte basso è 0, il controllo viene passato alla locazione $1875, il volume è azzerato ed il programma si interrompe.

Questo programma suona 19 note, alla 20esima viene letto uno zero come valore di frequenza del byte basso (alla locazione $18DC), così il volume viene posto a zero ed il programma finisce. Questo dato viene inserito come esempio. Ponete il vostro dato secondo la tabella delle note nella Figura 12-15 alla fine del capitolo e secondo la tabella della durata a pag. 93.

Estendete questo esempio. Aggiungete voci multiple usando diverse forme d'onda. Utilizzate il filtro per perfezionare la qualità delle note musicali. Usate questo programma come base per il vostro programma musicale completo.

SINCRONIZZAZIONE E MODULAZIONE AD ANELLO

Il chip SID del 6581 vi permette di creare strutture armoniche più complesse con la sincronizzazione o la modulazione ad anello di due voci.

Il procedimento di sincronizzazione è un'operazione AND (congiunzione) di due forme d'onda. Quando una delle due è zero, l'output è zero.

Ecco un algoritmo di sincronizzazione:

1 Azzerate i registri del suono del chip.
2 Ponete la frequenza alta per la voce 1.
3 Ponete Attack/Decay per la voce 1 (A=13, D=11).
4 Ponete la frequenza alta per la voce 3.
5 Ponete il volume 15.
6 Ponete l'inizio del triangolo, regolate la sincronizzazione della forma d'onda per la voce 1.
7 Ciclo di temporizzazione.
8 Azzerate il triangolo, regolate la sincronizzazione della forma d'onda per la voce 1.
9 Aspettate, poi spegnete il volume.

La funzione di sincronizzazione è abilitata (accesa) al passo 6, dove sono i bit 0, 1 e 4 del registro $D404. Il bit 1 abilita la funzione di sincronizzazione tra la voce 1 e la voce 3. I bit 0 e 4 hanno le loro solite funzioni di eseguire il gate della voce 1 e di porre la forma d'onda triangolare.

La modulazione del suono (compiuta per la voce 1 ponendo il bit 2 del registro $D404 al passo 6 dell'algoritmo) pone nuovamente l'output triangolo dell'oscillatore 1 con una combinazione "modulata del suono" degli oscillatori 1 e 3. Questo produce strutture di ipertoni non armoniche utilizzate per imitare i suoni della campana o del gong.

DESCRIZIONE DEL REGISTRO SID

Segue una descrizione registro per registro del chip SID in ordine numerico.

VOCE 1

FREQ BASSA/FREQ ALTA (Registri 00, 01)

Questi registri formano insieme un numero a 16 bit che controlla linearmente la frequenza dell'oscillatore 1. La frequenza è determinata dall'equazione:

$F_{out} = (F_n \text{ per } F_{clk}/16777216)$ Hz

Dove F_n è il numero a 16 bit nei registri di Frequenza e F_{clk} è il clock del sistema applicato all'input Φ2 (pin 6). Per un clock standard 1.0-MHz, la frequenza è data da:

$F_{out} = (F_n \text{ per } 0.06097)$ Hz

Una tabella completa dei valori per generare otto ottave della scala musicale modulata normalmente dell'insieme A (440 Hz) è fornita nella Figura 12-15. Si nota che la risoluzione della frequenza del SID è sufficiente per qualsiasi scala di accordi e permette lo sweep da nota a nota (portamento) senza passi di frequenza distinguibili.

PW BASSO/PW ALTO (Registri 02,03)

Questi registri insieme formano un numero a 12 bit (i bit 4-7 di PW HI (ALTO) non sono usati) che controlla linearmente la durata dell'Impulso (tempo di messa sotto tensione o duty cycle) della forma d'onda dell'Impulso sull'Oscillatore 1. La durata dell'impulso è determinata dalla seguente equazione:

$PW_{out} = (PW_n/40.95)$ %

Dove PW_n è il numero a 12 bit nei registri della durata (ampiezza) dell'Impulso.

La risoluzione della durata dell'impulso fa sí che l'ampiezza compia lo sweep dolcemente senza passaggi percepibili. Notate che la forma d'onda dell'Impulso sull'Oscillatore 1 deve essere selezionata perchè i registri della ampiezza dell'Impulso abbiano un effetto udibile. Un valore di 0 o 4095 ($FFF) nei registri dell'ampiezza dell'Impulso produrrà un output costante DC, mentre un valore di 2048 ($800) produrrà un'onda quadra.

REGISTRO DI CONTROLLO (Registro 04)

Questo registro contiene otto bit di controllo che selezionano varie opzioni sull'Oscillatore 1.

GATE (bit 0): Il bit GATE controlla il Generatore Envelope per la Voce 1. Quando questo bit è risettato a 1, il Generatore Envelope subisce il Gate (è fatto scattare) ed il ciclo ATTACK/DECAY/SUSTAIN è inizializzato. Quando il bit è posto nuovamente a zero, comincia il ciclo RELEASE. Il Generatore Envelope controlla l'estensione dell'Oscillatore 1 che appare in output audio, perciò, il bit GATE deve essere posto (insieme ai parametri envelope appropriati) perchè l'output audio selezionato dell'Oscillatore 1 sia udibile. Una trattazione dettagliata del Generatore Envelope si può trovare nel settimo volume.

SINCRONIZZAZIONE (SYNC) (Bit 1): Il bit SYNC, se posto a 1, sincronizza la frequenza fondamentale dell'Oscillatore 1 con la frequenza fondamentale dell'Oscillatore 3, producendo effetti di "Hard Sync".

Variando la frequenza dell'Oscillatore 1 rispetto all'Oscillatore 3 si produce una larga scala di strutture armoniche complesse dalla Voce 1 alla frequenza dell'Oscillatore 3. Perchè avvenga la sincronizzazione, l'Oscillatore 3 deve essere posto ad una frequenza diversa da zero ma preferibilmente minore della frequenza dell'Oscillatore 1. Nessun altro parametro della Voce 3 ha effetto sulla sincronizzazione.

RING MOD (Bit 2): Il bit RING MOD, se posto a 1, pone nuovamente l'output della forma d'onda Triangolo dell'Oscillatore 1 con una combinazione degli Oscillatori 1 e 3 "Ring Modulated". Variando la frequenza dell'Oscillatore 1 rispetto all'Oscillatore 3 si produce una larga scala di strutture di ipertoni non armonici per creare i suoni della campana o del gong e per gli effetti speciali. Perchè la modulazione ad anello sia udibile, deve essere selezionata la forma d'onda Triangolo dell'Oscillatore 1 e l'Oscillatore 3 deve essere posto ad una frequenza diversa da zero. Nessun altro parametro della Voce 3 ha effetto sulla modulazione ad anello.

TEST (Bit 3): Il bit TEST, se posto a 1, risetta e blocca l'Oscillatore 1 a zero finchè il bit TEST viene azzerato. Anche l'output della forma d'onda Rumore dell'Oscillatore 1 viene risettata e l'output della forma d'onda Impulso è tenuta ad un livello DC. Normalmente, questo bit è usato per testaggio, tuttavia, può essere usato per sincronizzare l'Oscillatore 1 con segnali esterni, mettendo la generazione di forme d'onda molto complesse sotto il controllo del software in tempo reale.

(Bit 4): Se posto a 1, viene selezionato l'output della forma d'onda Triangolo dell'Oscillatore 1. La forma d'onda Triangolo è bassa di armoniche ed è dolce, come un flauto.

(Bit 5): Se posto a 1, viene selezionato l'output della forma d'onda Dente di Sega dell'Oscillatore 1. La forma d'onda Dente di Sega è ricca di armoniche dispari e pari ed è acuta come un ottone.

(Bit 6): Se posto a 1, viene selezionato l'output della forma d'onda Impulsi dell'Oscillatore 1. Il contenuto delle armoniche di questa forma d'onda può essere regolato dai registri di ampiezza dell'Impulso, che producono qualità del tono che vanno da un'onda quadra acuta e cupa ad un impulso nasale e stridulo. Eseguendo lo sweep dell'ampiezza dell'impulso in tempo reale si produce un effetto dinamico "graduale" che dà un senso di moto al suono. Saltando rapidamente tra diverse ampiezze di impulso si possono produrre interessanti sequenze di suoni armonici.

NOISE (RUMORE) (Bit 7): Se posto a 1, viene selezionato l'output della forma d'onda Rumore dell'Oscillatore 1. Questo output è un segnale casuale che cambia alla frequenza dell'Oscillatore 1. La qualità del suono può variare da un rombo basso ad un rumore bianco sibilante attraverso i registri di Frequenza dell'Oscillatore 1. Il rumore è utile per creare esplosioni, colpi di arma da fuoco, motori di jet, vento, onde ed altri suoni non intonati, sia tamburi drum che cimbali. Eseguendo lo sweep della frequenza dell'oscillatore, avendo selezionato il rumore, si produce un effetto drammatico forzato.

Si deve selezionare un output delle forme d'onda perchè l'Oscillatore 1 sia udibile, tuttavia NON è necessario eliminare la selezione delle forme d'onda per far cessare l'output della Voce 1. L'ampiezza della Voce 1 nell'output finale è solo una funzione del Generatore Envelope.

NOTA: Gli output delle forme d'onda dell'oscillatore NON sono aggiuntivi. Se si seleziona più di un output delle forme d'onda contemporaneamente, il risultato sarà una congiunzione logica delle forme d'onda. Sebbene questa tecnica possa essere usata per generare forme d'onda aggiuntive oltre le quattro elencate, deve essere utilizzata con attenzione. Se viene selezionata un'altra forma d'onda qualsiasi mentre è attiva quella del rumore, l'output del rumore si può "bloccare". Se questo accade, l'output rumore rimarrà in silenzio finchè sarà risettato dal bit TEST o sarà posto RES (pin 5) basso.

ATTACK/DECAY (Registro 05)

I bit 4-7 di questo registro (ATK0-ATK3) selezionano 1 dei 16 ritmi ATTACK per il Generatore Envelope della Voce 1. ATTACK determina la rapidità con la quale l'output della Voce 1 si alza da zero al massimo della ampiezza quando il Generatore Envelope subisce il Gate. I 16 ritmi ATTACK sono listati nella Figura 12-14.

I bit 0-3 (DCY0-DCY3) selezionano 1 dei 16 ritmi DECAY per il Generatore Envelope. Il ciclo DECAY segue il ciclo ATTACK ed i ritmi DECAY determinano la rapidità della caduta dell'output dal massimo della ampiezza al livello SUSTAIN selezionato. I 16 ritmi DECAY sono listati nella Figura 12-14.

SUSTAIN/RELEASE (Registro 06)

I bit 4-7 di questo registro (STN0-STN3) selezionano 1 dei 16 livelli SUSTAIN per il Generatore Envelope. Il ciclo SUSTAIN segue il ciclo DECAY e l'output della Voce 1 rimarrà alla ampiezza SUSTAIN selezionata per tutto il tempo in cui il bit gate ha valore. I livelli SUSTAIN vanno da zero al massimo della ampiezza in 16 passi lineari, con un valore SUSTAIN di 0 che seleziona la ampiezza zero ed un valore SUSTAIN di 15 ($F) che seleziona il massimo della ampiezza. Un valore SUSTAIN di 8 causerebbe il SUSTAIN della Voce 1 ad una ampiezza di metà del massimo della ampiezza raggiunta dal ciclo ATTACK.

I bit 0-3 (RLS0-RLS3) selezionano 1 dei 16 ritmi RELEASE per il Generatore Envelope. Il ciclo RELEASE segue il ciclo SUSTAIN quando il bit Gate è posto nuovamente a zero. A questo punto, l'output della Voce 1 cadrà dalla ampiezza SUSTAIN alla ampiezza zero del ritmo RELEASE selezionato. I 16 ritmi RELEASE sono identici ai ritmi DECAY come listati nella Figura 12-14.

NOTA: Il ciclo del Generatore Envelope può essere modificato in qualsiasi punto con il bit gate. Il Generatore Envelope può subire il Gate o il Release senza restrizioni. Per esempio, se il bit gate viene posto nuovamente prima che l'envelope abbia terminato il ciclo ATTACK, il ciclo RELEASE comincerà immediatamente, cominciando da qualsiasi ampiezza sia stata raggiunta. Se l'envelope subisce ancora il gate (prima che il ciclo RELEASE abbia raggiunto la ampiezza zero), comincerà un altro ciclo ATTACK partendo da qualsiasi ampiezza sia stata raggiunta. Questa tecnica può essere usata per generare envelope di ampiezza complessa con il controllo del software in tempo reale.

VALORE		RITMOATTACK	RITMODECAY/RELEASE
DEC.	(ESA)	(TEMPO/CICLO)	(TEMPO/CICLO)
0	(0)	2 ms	6 ms
1	(1)	8 ms	24 ms
2	(2)	16 ms	48 ms
3	(3)	24 ms	72 ms
4	(4)	38 ms	114 ms
5	(5)	56 ms	168 ms
6	(6)	68 ms	204 ms
7	(7)	80 ms	240 ms
8	(8)	100 ms	300 ms
9	(9)	250 ms	750 ms
10	(A)	500 ms	1.5 s
11	(B)	800 ms	2.4 s
12	(C)	1 s	3 s
13	(D)	3 s	9 s
14	(E)	5 s	15 s
15	(F)	8 s	24 s

Figura 12-14. Indici Envelope

NOTA: I ritmi envelope sono basati su un clock 1.0 MHz Φ2. Per altre frequenze Φ2, moltiplicate il ritmo dato per 1 MHz/ Φ2. I ritmi si riferiscono alla quantità di tempo per ciclo. Per esempio, dato un valore ATTACK di 2, il ciclo ATTACK impiegherà 16 ms per salire da zero al massimo della ampiezza. I ritmi DECAY/RELEASE fanno riferimento alla quantità di tempo che questi cicli impiegano per cadere dal massimo della ampiezza a zero.

VOCE 2

I registri $07-$0D controllano la Voce 2 e sono identici per funzioni ai registri $00-$06 con queste eccezioni:

1. Se selezionato, SYNC sincronizza l'Oscillatore 2 con l'Oscillatore 1.
2. Se selezionato, RING MOD pone nuovamente l'output Triangolo dell'Oscillatore 2 con la combinazione di suono modulato degli Oscillatori 2 e 1.

VOCE 3

I registri $0E-$14 controllano la Voce 3 e sono identici per funzioni ai registri $00-$06 con queste eccezioni:

1. Se selezionato, SYNC sincronizza l'Oscillatore 3 con l'Oscillatore 2.
2. Se selezionato, RING MOD pone nuovamente l'output Triangolo dell'Oscillatore 3 con la combinazione del suono modulato degli Oscillatori 3 e 2.

Un'operazione tipica di una voce consiste nel selezionare i parametri desiderati: frequenza, forma d'onda, effetti (SYNC, RING MOD) ed i ritmi envelope, poi nel fare il gate della voce tutte le volte che si desidera il suono. Il suono può essere mantenuto per qualsiasi lunghezza di tempo e fatto finire azzerando il bit gate. Ogni voce può venire usata separatamente, con parametri e gate indipendenti, o insieme per creare una voce sola e potente. Se sono usati insieme, una leggera discordanza di ogni oscillatore o accordanza di intervalli musicali crea un suono ricco e vivace.

FILTRO

FC BASSO/FC ALTO (REGISTRI $15, $16)

Questi registri insieme formano un numero a 11 bit (i bit 3-7 di FC LO (BASSO) non sono usati) che controlla letteralmente la Frequenza di Cutoff (o di Centro) del Filtro programmabile. La Frequenza di Cutoff approssimata va da 30 Hz a 12 KHz.

RES/FILT (REGISTRO $17)

I bit 4-7 di questo registro (RES0-RES3) controllano la risonanza del filtro. La risonanza è un effetto che enfatizza le componenti della frequenza uguali alla Frequenza di Cutoff del Filtro, causando un suono più alto.
Ci sono sedici impostazioni di risonanza che vanno in modo lineare dalla non risonanza (0) al massimo della risonanza (15 o $F). I bit 0-3 determinano quali segnali saranno istradati attraverso il Filtro:

FILT 1 (Bit 0): Se posto a zero, la Voce 1 appare direttamente in output audio ed il Filtro non ha effetto su di essa. Se posto a 1, la Voce 1 passerà attraverso il Filtro ed i contenuti armonici della Voce 1 saranno alterati secondo i parametri selezionati del Filtro.

FILT 2 (Bit 1): Lo stesso del bit 0 per la Voce 2.
FILT 3 (Bit 2): Lo stesso del bit 0 per la Voce 3.
FILTEX (Bit 3): Lo stesso del bit 0 per l'input audio Esterno (pin 26).

MODE/VOL (REGISTRO $18)

I bit 4-7 per questo registro selezionano vari modi Filtro ed opzioni output:

LP (Bit 4): Se posto a 1, viene selezionato l'output Passa Basso del Filtro e mandato in output audio. Per un dato segnale input del Filtro, tutte le componenti della frequenza sotto la Frequenza di Cutoff del Filtro vengono fatte passare inalterate, mentre tutte le componenti sopra quella di Cutoff vengono attenuate con un ritmo di 12 dB/Ottava. Il modo Passa Basso produce suoni corposi.

BP (Bit 5): Lo stesso del bit 4 per l'output Passa Banda. Tutte le componenti delle frequenze sopra e sotto quella di Cutoff sono attenuate con un ritmo di 6 dB/Ottava. Il modo Passa Banda produce suoni fievoli ed aperti.

HP (Bit 6): Lo stesso del bit 4 per l'output Passa Alto. Tutte le componenti delle frequenze sopra quella di Cutoff vengono passate inalterate, mentre tutte le componenti delle frequenze sotto quella di Cutoff sono attenuate con un ritmo di 12 dB/Ottava. Il modo Passa Alto produce suoni metallici e ronzanti.

3 OFF (Bit 7): Se posto a 1, l'output della Voce 3 è disinserito dal percorso diretto dell'audio. Ponendo la Voce 3 per oltrepassare il Filtro (FILT 3 = 0) e ponendo 3 OFF a 1, si fa sí che la Voce 3 non raggiunga l'output audio. Questo permette alla Voce 3 di essere usata per la modulazione senza alcun output indesiderato.

NOTA: I modi output del Filtro SONO aggiuntivi e si possono selezionare contemporaneamente modi del Filtro multipli. Per esempio, sia il modo LP che quello HP possono essere selezionati per produrre una risposta di Filtro a Tacca (o Eliminatore di Banda). Perchè il Filtro abbia un effetto udibile, deve essere selezionato almeno un output del Filtro ed almeno una Voce deve essere istradata attraverso il Filtro. Il Filtro è, forse, l'elemento più importante nel SID poichè permette che vengano generati toni complessi attraverso una sintesi sottrattiva (il Filtro viene usato per eliminare specifiche componenti delle frequenze da un segnale di input armonicamente ricco). Si ottengono i migliori risultati variando la Frequenza di Cutoff in tempo reale.

Bit 0-3 (VOL0-VOL3) selezionano uno dei 16 livelli fondamentali del Volume per l'output audio finale composto. I livelli dell'output del volume vanno da no output (0) al massimo del volume (15 o $F) in sedici passi lineari. Questo controllo può essere usato come un controllo statico del volume per bilanciare i livelli nei sistemi multichip o per creare effetti dinamici di volume, come il Tremolo. Deve essere selezionato un livello di Volume diverso da zero perchè il SID produca un suono.

MISCELLANEO

POTX (REGISTRO $19)

Questo registro permette al microprocessore di leggere la posizione del potenziometro legato a POTX (pin 24), con i valori che vanno da 0 resistenza minima a 255 ($FF) resistenza massima. Il valore è sempre valido ed è aggiornato ad ogni ciclo della clock 512 Φ2. Per informazioni sui valori del potenziometro e del condensatore, si veda il settimo volume, capitolo 17 sezione Descrizione dei Pin del SID.

POTY (REGISTRO $IA)

Lo stesso del POTX per il potenziometro (pot) legato a POTY (pin 23).

OSC 3/RANDOM (REGISTRO $IB)

Questo registro permette al microprocessore di leggere gli 8 bit superiori dell'output dell'Oscillatore 3. Il carattere dei numeri generati è in relazione diretta con la forma d'onda selezionata. Se è selezionata la forma d'onda Dente di Sega dell'Oscillatore 3, il registro presenterà una serie di numeri incrementandoli da 0 a 255 ($FF) con una velocità determinata dalla frequenza dell'Oscillatore 3. Se viene selezionata la forma d'onda Triangolo, l'output incrementerà da 0 a 255, poi decrementerà fino a 0. Se viene selezionata la forma d'onda Impulso, l'output salterà tra 0 e 255. Selezionando la forma d'onda Rumore si produrranno una serie di numeri casuali (random), perciò questo registro può essere usato come generatore di numeri casuali per i giochi. Ci sono numerose applicazioni di misurazione del tempo e di sequenziamento per il registro OSC 3; ma la funzione principale è probabilmente quella di un generatore di modulazione. I numeri generati da questo registro possono essere sommati, via software, ai registri dell'Oscillatore o della Frequenza del Filtro o ai registri di Ampiezza dell'Impulso in tempo reale. Con questo metodo si possono generare molti effetti dinamici. I suoni tipo sirena possono essere creati aggiungendo l'output Dente di Sega dell'OSC 3 al controllo della frequenza di un altro oscillatore. Effetti del sintetizzatore "Sample and Hold" (Analizza e Mantieni) possono essere prodotti aggiungendo l'output Rumore dell'OSC 3 ai registri di controllo della Frequenza del Filtro. Il Vibrato può essere prodotto ponendo l'Oscillatore 3 ad una frequenza attorno a 7 Hz ed aggiungendo l'output Triangolo dell'OSC 3 (con una graduazione adeguata) al controllo della Frequenza di un altro oscillatore. È disponibile una quantità illimitata di effetti alterando la frequenza dell'Oscillatore 3 e graduando l'output di OSC 3. Se l'Oscillatore 3 è usato per la modulazione, l'output audio della Voce 3 dovrebbe essere eliminato (3 OFF=1).

ENV 3 (Registro $IC)

Lo stesso di OSC 3, ma questo registro permette al microprocessore di leggere l'output del Generatore Envelope della Voce 3. Questo output può essere aggiunto alla Frequenza del Filtro per produrre envelope armonici, WAH-WAH, ed effetti simili. Suoni "phaser" (di fase) possono essere creati aggiungendo questo output ai registri di controllo della frequenza di un oscillatore. Il Generatore Envelope della Voce 3 deve subire il Gate per produrre qualsiasi output da questo registro. Il registro OSC 3 riflette sempre il cambiamento dell'output dell'oscillatore e non è toccato in alcun modo dal Generatore Envelope.

VALORI DELLA SCALA MUSICALE A TEMPERAMENTO EQUABILE

La tabella della Figura 12-15 lista i valori numerici che devono essere memorizzati nei registri di controllo della frequenza dell'Oscillatore SID per produrre le note della scala musicale a temperamento equabile. La scala a temperamento equabile consiste in un'ottava contenente 12 semitoni (note): C (Do), D (Re), E (Mi), F (Fa), G (Sol), A (La), B (Si), e C# (Do#), D# (Re#), F# (Fa#), G# (Sol#), A# (La#). La frequenza di ogni semitono è esattamente la 12esima radice di 2 ($\sqrt[12]{2}$) volte la frequenza del semitono precedente. La tabella è basata su una clock Φ2 di 1.02 MHz. Riferitevi all'equazione data nella Descrizione dei Registri per l'uso di frequenze di altri clock master. La scala selezionata è un insieme di toni, nella quale A-4 = 440 Hz. Sono anche possibili trasposizioni di questa scala e di scale diverse dalla scala a temperamento equabile.

Sebbene la tabella della Figura 12-15 fornisca un metodo veloce e semplice per generare una scala a temperamento equabile, non è molto efficiente per quanto riguarda la memoria poichè richiede 192 byte solo per la tabella. L'efficienza della memoria può essere aumentata determinando il valore delle note con un algoritmo. Utilizzando il concetto che ogni nota in un'ottava è esattamente metà della frequenza di quella nota nell'ottava successiva, la tabella migliorata delle note può essere ridotta da 96 elementi a 12 elementi, poichè ci sono 12 note per ottava. Se i 12 elementi (24 byte) consistono dei valori a 16 bit per l'8ava ottava (C-7 fino a B-7), allora le note nelle ottave più basse possono essere derivate scegliendo la nota appropriata dell'8ava ottava e dividendo il valore a 16 bit per 2, per ogni ottava di differenza. Poichè la divisione per due non è altro che uno spostamento a destra del valore, il calcolo può facilmente essere compiuto da una semplice routine di software. Sebbene la nota B-7 sia oltre l'intervallo degli oscillatori, questo valore dovrebbe essere incluso ancora nella tabella per i calcoli (l'MSB di B-7 richiederà uno speciale caso di software, per generare questo bit nel CARRY prima di operare lo SHIFT). Ogni nota deve essere specificata in una forma che indichi quale dei 12 semitoni si desidera, ed in quale delle 8 ottave è il semitono. Poichè sono necessari 4 bit per selezionare uno dei 12 semitoni e 3 bit sono necessari per selezionare una delle 8 ottave, l'informazione può collocarsi in un byte, con il nybble più basso che seleziona il semitono (indirizzando la tabella migliorata) ed il nybble più alto che viene usato per la routine della divisione per determinare quante volte il valore della tabella deve essere spostato a sinistra.

VALORI DELLE NOTE MUSICALI

La Figura 12-15 contiene una lista completa delle Note #, delle note reali e dei valori che devono essere memorizzati nei registri HI FREQ e LOW FREQ del chip del suono per produrre le note indicate.

Figura 12-15. Valori delle Note Musicali

NOTA	MUSICALE	FREQ OSCILLATORE		
NOTA	OTTAVA	DECIMALE	ALTO	BASSO
0	C-0	268	1	12
1	C#-0	284	1	28
2	D-0	301	1	45
3	D#-0	318	1	62
4	E-0	337	1	81
5	F-0	358	1	102
6	F#-0	379	1	123
7	G-0	401	1	145
8	G#-0	425	1	169
9	A-0	451	1	195
10	A#-0	477	1	221
11	B-0	506	1	250
16	C-1	536	2	24
17	C#-1	568	2	56
18	D-1	602	2	90
19	D#-1	637	2	125
20	E-1	675	2	163
21	F-1	716	2	204
22	F#-1	758	2	246
23	G-1	803	3	35
24	G#-1	851	3	83
25	A-1	902	3	134
26	A#-1	955	3	187
27	B-1	1012	3	244
32	C-2	1072	4	48
33	A#-2	1136	4	112
34	D-2	1204	4	180
35	D#-2	1275	4	251
36	E-2	1351	5	71
37	F-2	1432	5	152
38	F#-2	1517	5	237
39	G-2	1607	6	71
40	G#-2	1703	6	167
41	A-2	1804	7	12
42	A#-2	1911	7	119
43	B-2	2025	7	233

Figura 12-15. Valori delle Note Musicali (continuazione)

NOTA MUSICALE		FREQ OSCILLATORE		
NOTA	OTTAVA	DECIMALE	ALTO	BASSO
48	C-3	2145	8	97
49	C#-3	2273	8	225
50	D-3	2408	9	104
51	D#-3	2551	9	247
52	E-3	2703	10	143
53	F-3	2864	11	48
54	F#-3	3034	11	218
55	G-3	3215	12	143
56	G#-3	3406	13	78
57	A-3	3608	14	24
58	A#-3	3823	14	239
59	B-3	4050	15	210
64	C-4	4291	16	195
65	C#-4	4547	17	195
66	D-4	4817	18	209
67	D#-4	5103	19	239
68	E-4	5407	21	31
69	F-4	5728	22	96
70	F#-4	6069	23	181
71	G-4	6430	25	30
72	G#-4	6812	26	156
73	A-4	7217	28	49
74	A#-4	7647	29	223
75	B-4	8101	31	165
80	C-5	8583	33	135
81	C#-5	9094	35	134
82	C-0	9634	37	162
83	C#-0	10207	29	223
84	D-0	10814	42	62
85	F-5	11457	44	193
86	F#-5	12139	47	107
87	G-5	12860	50	60
88	G#-5	13625	53	57
89	A-5	14435	56	99
90	A#-5	15294	59	190
91	B-5	16203	63	75
96	C-6	17167	67	15
97	C#-6	18188	71	12
98	D-6	19269	75	69
99	D#-6	20415	79	191
100	E-6	21629	84	125
101	F-6	22915	89	131
102	F#-6	24278	94	214
103	G-6	25721	100	121
104	G#-6	27251	106	115
105	A-6	28871	112	199
106	A#-6	30588	119	124
107	B-6	32407	126	151
112	C-7	34334	134	30

Figura 12-15. Valori delle Note Musicali (continuazione)

NOTA MUSICALE		FREQ OSCILLATORE		
NOTA	OTTAVA	DECIMALE	ALTO	BASSO
113	C#-7	36376	142	24
114	D-7	38539	150	139
115	D#-7	40830	159	126
116	E-7	43258	168	250
117	F-7	45830	179	6
118	F#-7	48556	189	172
119	G-7	51443	200	243
120	G#-7	54502	212	230
121	A-7	57743	225	143
122	A#-7	61176	238	248
123	B-7	64814	253	46

IMPOSTAZIONI DEL FILTRO

LOCAZIONE	CONTENUTI
54293	Frequenza di cutoff bassa (0-7)
54294	Frequenza di cutoff alta (0-255)
54295	Risonanza (bit 4-7)
	Filtro della Voce 3(bit2)
	Filtro della Voce 2 (bit 1)
	Filtro della Voce 1 (bit 0)
54296	Passa Alto (bit 6)
	Passa Banda (bit 5)
	Passa Basso (bit 4)
	Volume (bit 0-3)

13

GUIDA ALL'INPUT/OUTPUT

INTRODUZIONE

Oltre ad essere un calcolatore ed un manipolatore di dati, il Commodore 128 è un trasmettitore. Attraverso le varie porte di connessione, può trasmettere e ricevere informazioni a/da diversi tipi di impianti periferici. Per la loro varietà, ogni apparecchiatura, deve avere il proprio codice unico di accesso e specifiche istruzioni (software). Questo capitolo include i dettagli di ogni collegamento e la tecnica di comunicazione con le varie apparecchiature ad essi associata.

PORTE DI INPUT/OUTPUT

La maggior parte dei collegamenti sono considerati di input e output perchè l'informazione viene trasmessa in entrambe le direzioni. Questi collegamenti includono la Porta Seriale del Commodore per il/i drive e la/le stampante/i, l'Ingresso dell'Utente per le comunicazioni esterne con i modem, un ingresso di espansione per un'ulteriore memoria, un connettore Datassette per un apparecchio di memorizzazione su nastro, due ingressi del monitor sia per il video composto che per i monitor RGBI, un connettore per la televisione, ed ingressi per i giochi per varie apparecchiature di controllo come i joystick, paddle e tavole grafiche. I dettagli sull'ingresso pinout iniziano più avanti, in questo capitolo.

DISK DRIVE (UNITÀ A DISCHI)

Il disk drive è il dispositivo di memorizzazione delle informazioni oggi più popolare. Esso memorizza i programmi ed i dati come segnali magnetici su un floppy disk di plastica flessibile coperto di ossido di ferro. Le informazioni (caratteri, lettere e numeri) rappresentate da una serie di numeri binari sono impresse magneticamente sul dischetto in una delle 35 tracce concentriche circolari. Ogni anello è diviso in settori ognuno da 17 a 21. Ogni anello può memorizzare circa 5000 byte per traccia. Un disk drive 1541 ha una capacità di circa 170K. Il disk drive 1571 ha doppia faccia, così un dischetto formattato nel 1571 tiene il doppio dei dati, circa 340K.

FORMATTAZIONE DI UN DISCHETTO

I dischetti vuoti nuovi di fabbrica sono senza tracce o settori. Un dischetto deve essere formattato dal drive del vostro computer secondo il sistema operativo del dischetto. Non potete memorizzare niente su un dischetto nuovo di zecca finchè questo processo, chiamato Formattazione o newing (rinnovamento) è completato. Il drive del Commodore ha dei microprocessori incorporati che organizzano un dischetto vuoto in modo appropriato, se vengono date loro le istruzioni dal computer. La 18esima traccia è usata come directory per il drive. Gli indirizzi di tutti i blocchi (oltre 600) sono memorizzati alla traccia 18 come Block Allocation Map (Mappa di Allocazione del Blocco) o più semplicemente come directory.

Prima di formattare un dischetto, decidete un nome di identificazione ed un codice di due cifre. Selezionate un nome fino a 16 caratteri che vi aiuterà ad identificare il vostro dischetto. Per esempio, il nome del dischetto può essere il vostro nome, od un argomento come i giochi. Il codice di due cifre può essere qualsiasi combinazione di lettera-numero. Questo codice avverte il drive immediatamente tutte le volte che cambiate il dischetto, conviene così cercare di assegnare ad ogni dischetto un codice unico. Il comando di formattazione per il modo C128 è il seguente:

```
HEADER "nome del dischetto", Ixy, D0, ON U8
```

Per il modo C64 o C128, usate:

```
OPEN 15, 8, 15, "N0:nome del dischetto, xy"
```

Notate che xy è un qualsiasi codice di due cifre che voi selezionate, come 01, 45, W3, 40 o WD. Lo 0 dopo la D o la N rappresenta il numero di default del drive per un singolo drive o il primo drive di una unità con due drive dove entrambi condividono un numero comune di dispositivo. L'8 in ogni frase rappresenta il numero del dispositivo. Un secondo drive sarebbe posto col numero 9, un terzo drive col numero 10, etc. Con un solo drive, il comando HEADER può essere semplicemente:

```
HEADER "nome del dischetto", Ixy
```

Il procedimento di formattazione richiede di solito circa 80 secondi nel drive 1541 e molto meno nel drive 1571. Una volta che questa operazione è completata, il dischetto può essere usato per memorizzare programmi e dati. Non formattatelo ancora, altrimenti tutto ciò che è salvato sarà cancellato. Per "pulire" semplicemente un disco già formattato, usate le istruzioni di cui sopra ma non includete la virgola seguita da un numero di identificazione. Questo manterrà il formato originale, userà il nome nuovo e cancellerà la directory, un procedimento considerevolmente più veloce della formattazione totale.

Il comando OPEN stabilisce una linea di comunicazione tra il computer ed un numero specifico del dispositivo. I drive sono predisposti dalla ditta con il numero di dispositivo 8 e possono essere convertiti al dispositivo 9, 10, 11 o 12. Consultate il manuale del vostro drive per i dettagli.

Per verificare di avere formattato in modo corretto il vostro dischetto, consultate la sezione sulla lettura del directory.

SALVATAGGIO DEI PROGRAMMI

Il procedimento di salvataggio per memorizzare i programmi su un dischetto non rimuove il programma dal computer. "Copia" semplicemente il programma od il file dal computer al dischetto nel drive. Non viene perduto nulla nel procedimento. I programmi in BASIC vengono salvati su un dischetto formattato con i seguenti comandi:

C128 o C64	SAVE "nome del programma", 8
solo C128	DSAVE "nome del programma"
	DSAVE "nome del programma", D0, U9

(se il drive ha come numero di dispositivo 9).

In modo C128, i programmi in linguaggio macchina o i file binari possono essere salvati con questi comandi BASIC:

BSAVE "nome del file", Bb, Ps, TO Pe
o BSAVE "nome del file", D0, U9, Bb, Ps TO Pe
(se il drive ha come numero di dispositivo 9).

In questi formati:

"b" è il numero del banco di memoria (da 0 a 15)
"s" è l'indirizzo di partenza
"e" è l'indirizzo finale

ESEMPIO:

BSAVE "nome del file", B2, P3584 TO P4096

Questo salva il file binario di nome "nome del file" dal banco di memoria 2, nell'intervallo di memoria tra 3584 e 4096 decimali.

Il comando seguente salva un programma in BASIC sia in modo C128 che in modo C64:

SAVE "nome del file", 8

dove 8 rappresenta il numero del dispositivo.

RIPOSIZIONAMENTO DI UN FILE O DI UN PROGRAMMA

Quando risalvate un programma con lo stesso nome del file, dovete usare il segno @ nel formato del comando SAVE, come segue:

In BASIC 7.0: DSAVE "@ nome del programma"
In BASIC 2.0: SAVE "@0:nome del programma", 8

Il simbolo @ dice al computer di memorizzare questo programma al posto di quello con lo stesso nome. Senza questo simbolo, il drive non permetterà che il nuovo programma sia memorizzato. Notate che durante il procedimento di riposizionamento, viene salvato il nuovo programma prima, poi quello vecchio è automaticamente cancellato ed il directory è aggiornato. Usate **VERIFY** per accertarvi di salvare in modo corretto.

VERIFICA DI UN PROGRAMMA O DI UN FILE

Potete voler verificare che un programma sia stato salvato accuratamente sul dischetto. Il comando VERIFY confronta il programma in memoria con il programma sul dischetto. Se i programmi sono identici byte per byte, viene visualizzato un messaggio OK.
In BASIC 7.0, usate i seguenti:

DVERIFY "nome del file"

oppure

DVERIFY "nome del file", D0, U9

Il primo comando verifica un programma usando un singolo drive assegnato come unità 8, ed il secondo comando verifica un programma usando un solo drive assegnato come unità 9.
In BASIC 2.0, usate:

VERIFY "nome del file", 8

oppure

VERIFY "nome del file", 8, 1

(per verificare il programma nella locazione di memoria nella quale era stato salvato, principalmente programmi in linguaggio macchina).
Dopo che è stato salvato e verificato, il vostro programma è memorizzato sul vostro dischetto per un uso futuro.

IL DIRECTORY

Una volta che il dischetto è formattato, esso ha un directory o catalogo che lista i file memorizzati sul dischetto. Prima di caricare un programma potreste vedere il directory per scoprire esattamente cosa è memorizzato sul dischetto. Quando chiamate il directory, lo schermo visualizza il nome del dischetto ed il numero di identificazione seguiti da una lista di programmi o di file attualmente memorizzati. Naturalmente, un dischetto appena formattato visualizzerà solo il suo nome ed il suo numero. Per vedere il directory, usate i comandi listati qui sotto.

In BASIC 7.0 usate i seguenti:

DIRECTORY
o CATALOG
o CATALOG D0, U9
o DIRECTORY D0, U9 (per un drive doppio, numero di unità 9)

Il programma in memoria non viene toccato da questo comando.

In BASIC 2.0 usate il seguente comando:

LOAD "$0", 8:LIST

Questo comando cancella il programma residente in memoria. Lo 0 è facoltativo con un singolo drive.

Per stampare un hard copy (documento stampato) di un directory, dovete usare il metodo BASIC 2.0. Dovete prima aprire un canale verso la stampante. Ciò è trattato nella sezione Output verso una stampante.

RINTRACCIAMENTO DI PROGRAMMI O DI FILE DAI DISCHETTI

Una volta che avete memorizzato file o programmi su un dischetto, li rintracciate con una versione appropriata del comando LOAD.

In BASIC 7.0 usate il seguente comando:

DLOAD "nome del file"
o
DLOAD "nome del file", D0, U9
(per l'unità di drive 9, drive 0)

Per rintracciare un programma o un file binario salvato dal comando BSAVE o dal monitor di linguaggio macchina, usate il comando BASIC 7.0 seguente:

BLOAD "nome del file", Bb, Ps
o
BLOAD "nome del file", D0, U9, Bb, Ps
(per l'unità di drive 9, drive 0)

dove b è il numero del banco ed s l'indirizzo di partenza.
In BASIC 2.0 usate i seguenti formati del comando:

Per caricare un programma in BASIC: LOAD "nome del file", 8
Per caricare in linguaggio macchina: LOAD "nome del file", 8, 1
Il segno di rilocazione è un 1 o uno 0, che è il valore di default. L'1 dice al computer di LOAD (caricare) dal punto in cui è stato SAVEd (salvato).

CREAZIONE E MEMORIZZAZIONE DEI FILE

Un file è una serie di dati messi in relazione: caratteri, numeri e lettere. Qualsiasi cosa venga memorizzata su un dischetto è considerato un file. Un programma è un tipo specifico di file che può venire eseguito o fatto girare. I data-file sono file che contengono dati originali usati da un programma. Questi file sono classificati come sequenziali, relativi, dell'utente o casuali. Il directory lista tutti i file ed indica il tipo di file: PRG per programma, SEQ per file sequenziale, REL per file relativo e USR per i file dell'utente.

Un programma è un file con le istruzioni operative per il computer. Un file sequenziale è una serie di dati memorizzati in sequenza e che possono essere letti solo in ordine sequenziale. Un file relativo è un file memorizzato in ordine sequenziale ed a cui si accede direttamente in qualsiasi locazione usando un puntatore.

La creazione di un file sequenziale è inizializzata con una istruzione OPEN. Un file è aperto (OPENed) con la seguente informazione:

- Numero del file: da 0 a 255; 15 per comandi speciali e controllo degli errori.
- Numero del dispositivo: il default è 8 per un drive; 4 per una stampante (4-30 sono disponibili per bus seriali delle unità).
- Numero del canale: da 0 a 255; è la linea di comunicazione stabilita tra le unità (conosciuta anche come indirizzo secondario).

Informazioni ulteriori (optional) sono specificate anche da:

- Il numero del drive seguito da due punti: il default è 0 per un singolo drive, 1 è specificato per un drive doppio.
- Nome del file: il nome del file fino a 16 caratteri.
- Tipo di file: S = sequenziale, P = programma, R = relativo, U = dell'utente
- Operazioni: (R = lettura, W = scrittura)

Il comando seguente:

OPEN 111, 8, 12, "0:NOME, S, W"

apre il file numero 111 all'unità numero 8, usando il canale 12. Il file è aperto al drive di default 0, con il nome del file "NOME," come file sequenziale. La W specifica che avrà luogo un'operazione di scrittura. Per memorizzare il dato, esso viene "scritto" sul dischetto. Il dato viene rintracciato "leggendo" il dischetto.

Immettete, salvate e fate girare i seguenti programmi come routine campione per creare, salvare e leggere un file sequenziale. Consultate anche il manuale del vostro drive.

```
10 REM MEMORIZZAZIONE DATI
11 REM IL PROGRAMMA TERMINA QUANDO RICEVE IN RISPOSTA LA PAROLA "END"
20 OPEN 5,8,5,"ESEMPIO,S,W"
25 DO
30 INPUT"NOME DI UN AMICO";N$
40 PRINT#5,N$
50 IF N$="END"THEN CLOSE 5:END
60 LOOP
```

NOTA: In modo C128, il Commodore 128 ha un insieme di comandi di input/output. Questi comprendono i comandi DLOAD e DSAVE già descritti, e DOPEN, DCLOSE ed altri comandi. Consultate l'Enciclopedia BASIC 7.0 nel primo Volume, Capitolo 3 o il vostro manuale del drive per dettagli su uno qualsiasi di questi comandi.

La riga 20 dice al drive di aprire un file sequenziale per la scrittura del dato. La riga 30 ottiene l'informazione dalla tastiera richiedendola sullo schermo. L'istruzione PRINT# alla riga 40 specifica il numero del file dove si deve scrivere l'informazione. La riga 40 è l'istruzione che scrive realmente il dato nel canale e che crea i contenuti del file. La riga 50 colloca un elemento di riferimento riconoscibile alla fine del file- la parola "end" alla fine del file. La riga 50 chiude il file dopo che tutti i dati sono stati scritti.

Ecco un programma che legge i dati dal file sequenziale appena creato:

```
10 REM CARICAMENTO FILE
20 OPEN 6,8,6"ESEMPIO,S,R"
25 DO
30 INPUT#6,L$
40 IF L$="END"THEN CLOSE6:END
50 PRINT L$
60 LOOP
```

La riga 20 apre un canale verso il file sequenziale chiamato ESEMPIO per la lettura. Una volta che trova il dato fine, il file è chiuso in modo adeguato. Poichè

ogni dato viene letto usando l'istruzione INPUT#, viene stampato sullo schermo con una semplice istruzione PRINT. Qualsiasi file sequenziale nel quale tutte le operazioni di scrittura siano state completate e che sia già chiuso, può essere aperto solo come file di lettura; non ci si può scrivere a meno che venga specificata un'operazione APPEND o che venga usato il prefisso @.

Si può espandere la lunghezza dei file con il comando APPEND in BASIC 7.0 come segue:

```
APPEND#7, "ESEMPIO"
```

Questo comando apre il file chiamato "ESEMPIO". Qualsiasi dato mandato ad esso in una PRINT#7 sarà aggiunto. Qualsiasi numero di file può essere specificato finchè il numero del file PRINT coincide con il numero del file APPEND.

Una volta inviato l'intero dato, il canale viene chiuso con:

```
DCLOSE#7
```

In BASIC 2.0, usate questo comando:

```
OPEN 7, 8, 7, "0:ESEMPIO, A"
```

La lettera A permette alle successive istruzioni PRINT# di inserire i dati alla fine del file "Esempio". Questa tecnica in BASIC 2.0 richiede anche una frase CLOSE: CLOSE 7.

ALTRI FILE

Il BASIC del C128 comprende parecchi comandi di preparazione del drive compresi **SCRATCH, RENAME, CONCAT, COPY** e **COLLECT** ed inoltre un comando del file relativo **RECORD**. Notate che queste operazioni sono disponibili anche in modo C64 se combinate con una frase OPEN sul canale 15. Consultate la vostra Guida Enciclopedica del Sistema C128 o il manuale del drive per i dettagli.

Consultate il manuale del vostro drive per i dettagli sui file relativi, sui file random e sull'accesso diretto del programma al microprocessore del drive via canale 15.

CAMBIAMENTO DEL NUMERO DI UNITÀ DEL DRIVE

Il numero di dispositivo del drive di solito è 8, ma può diventare il 9, 10 o 11 con un cambiamento sia hardware che software. Il comando seguente modifica il vostro drive nell'unità 9 attraverso il software finchè non viene spento l'alimentatore:

```
OPEN 15, 8, 15:PRINT#15, "M—W" CHR$(119)CHR$(0)CHR$(2)
CHR$(41)CHR$(73):CLOSE 15
```

Il 1571 ha un commutatore del numero di unità sul retro.

OUTPUT VERSO UNA STAMPANTE

Il vostro Commodore 128 trasmette i dati attraverso la porta seriale a molti tipi diversi di stampanti. Le stampanti più popolari del Commodore sono i tipi margherita e matrice a punti. Il sistema margherita utilizza lettere modellate di metallo o di plastica simili a quelle di una macchina da scrivere e produce la stampa migliore. I denti che tengono questi caratteri sono montati su un sostegno centrale, e la ruota sembra una margherita. La testina della stampante fa girare la ruota finchè la lettera giusta è posizionata dietro ad un martelletto, che la preme contro un nastro sulla carta.

Il sistema più versatile è la matrice a punti. Usando una colonna di piedini di metallo, una serie di punti viene impressa sulla carta attraverso un nastro di inchiostro quando la testina della stampante si muove sulla pagina. La combinazione dei punti è basata sul codice binario di ogni carattere. Il numero infinito di potenziali combinazioni di punti rende possibile stampare qualsiasi tipo immaginabile di carattere dal corsivo al giapponese dipendendo ciò dal software interno (firmware) della stampante.

Sebbene molte stampanti dicano che il loro sistema sia compatibile con il Commodore 128, se la Commodore non ne autorizza l'uso, si deve avere una speciale interfaccia per la stampante. Le stampanti citate come parallele Centronics o Seriali non corrispondono direttamente a nessuna delle porte del Commodore. Il Commodore non è adatto a una stampante o interfacce di terzi. La Porta Seriale del Commodore è unica e non è uguale alla Porta Seriale RS232, come richiedono molte stampanti.

Il Commodore può visualizzare 512 caratteri. Il secondo insieme di 256 include sia le lettere maiuscole che quelle minuscole; il primo include solo quelle maiuscole con la grafica. Ciò è possibile usando codici identici a cui si accede separatamente. Di conseguenza, solo le stampanti Commodore a matrici a punti sono programmate per stampare tutti i caratteri disponibili sul Commodore. A causa di queste differenze, i codici ASCII del Commodore non sono uguali a quelli ASCII standard. Perciò vengono richieste modifiche speciali quando si usa l'interfaccia con la dotazione non-Commodore.

Per fare funzionare una stampante, deve essere aperta una linea di comunicazione con il computer usando l'istruzione OPEN. La stampante ha di solito il numero di dispositivo 4, sebbene spesso il commutatore sia selezionabile a 5. Il comando:

```
OPEN 3, 4
```

stabilisce un legame con la stampante con il numero di dispositivo 4. Il primo numero può essere un numero qualsiasi tra 1 e 255. I numeri oltre a 128 forzano un salto di linea dopo il termine della riga.

STAMPA DI UN LISTATO DI PROGRAMMA

I seguenti comandi produrranno un listato stampato:

```
OPEN3,4:CMD3:LIST
```

Il comando CMD ridirige tutti gli output dello schermo alla stampante. Qualsiasi output successivo viene inviato alla stampante finchè essa non è spenta con:

```
PRINT#3:CLOSE3
```

Queste istruzioni sono necessarie per pulire il canale e per assicurarsi che tutti i comandi siano stati inviati in modo appropriato alla stampante.

NOTA: Per stampare un listato con la spaziatura doppia, un line feed (salto di linea) può essere forzato usando un numero di file maggiore di 127 come:

```
OPEN131,4:CMD131:LIST:PRINT#131:CLOSE131
```

STAMPA DI UNA LINEA DIGITATA

Un comando PRINT# associato a un numero di file di stampa aperto mette in output il dato associato alla stampante. I seguenti comandi mettono in output tutto ciò che è digitato:

```
OPEN3,4:PRINT#3,"DIGITA QUALSIASI COSA"
```

Dopo aver premuto **RETURN,** la linea viene inviata alla stampante e tutto ciò che è tra virgolette viene stampato sulla carta.

STAMPA ATTRAVERSO UN PROGRAMMA

Le istruzioni indirizzate a una stampante richiedono una istruzione PRINT# e possono essere unite alla istruzione PRINT per la visualizzazione. Il programma seguente stampa cinque linee di testo per ogni frase PRINT# mentre usa la visualizzazione dello schermo per ottenere l'input:

```
10 OPEN 3,4,7
20 DO UNTIL L=5
25 IF L=0THEN30:ELSE INPUTA$:GOTO40
30 INPUT"COSA VUOI SCRIVERE";A$
40 PRINT#3,A$
50 L=L+1
60 LOOP
70 CLOSE3
```

Questo è l'inizio di un semplice word processor e si potrebbe estendere in un sistema complesso con il controllo della stampante.

GESTIONE DELLA VOSTRA STAMPANTE

Ogni possibilità della stampante (come lettere minuscole, line feed, caratteri a doppia larghezza, caratteri inversi, grafici a punti, e caratteri speciali) è gestita inviando l'appropriato codice di controllo dal computer. A seconda della stampante che si usa, ci sono due modi di invio del codice: con la funzione CHR$ o con l'indirizzo secondario.

I codici standard vengono inviati con la funzione BASIC CHR$(X). I codici ASCII sotto il 27 sono riservati generalmente ai codici della stampante (come 14 per la funzione di doppia larghezza). I codici sopra il 27 rappresentano specifici caratteri della tastiera. I codici di gestione della stampante sopra il 27 (per esempio 145) sono preceduti da ESCape codice 27. Alcuni sistemi accetteranno CHR$(27) seguito da CHR$(X) o dal carattere equivalente tra virgolette. Così il carattere in grassetto rappresentato dal Codice 33 può essere acceso con:

```
PRINT#3, CHR$(27)CHR$(33)
o PRINT#3, CHR$(27)"!"
```

e spento con:

```
PRINT#3, CHR$(27)CHR$(34)
```

dando per scontato che il canale 3 fosse aperto in precedenza.

Molti word processor possono procurare speciali gestioni delle stampanti con almeno uno di questi metodi. Consultate i vostri manuali della stampante e del word processor per la tecnica di invio dei caratteri di controllo alla vostra stampante.

Il metodo alternativo di invio dei codici di controllo ad una stampante è con l'indirizzo secondario nella frase OPEN. Una istruzione OPEN senza un indirizzo secondario porta direttamente a 0. Per questo OPEN3,4 è uguale a OPEN3,4,0. Generalmente un indirizzo secondario di 7 farà sì che la stampante stampi con caratteri maiuscoli e minuscoli:

```
OPEN3,4,7
```

Altri indirizzi secondari sono spesso disponibili ed eseguono varie operazioni in conformità con la struttura della vostra stampante. Consultate il vostro manuale per specificazioni. La maggior parte dei word processor permettono l'uso degli indirizzi secondari come codici incorporati.

Il programma seguente aprirà un canale verso un file sequenziale sul dischetto, leggerà i dati e li metterà in output verso una stampante:

```
10 OPEN 3,8,3,"ESEMPIO,S,R"
20 OPEN 4,4,0
25 DO
30 INPUT#3,L$
40 IF L$="END"THEN80ELSE50
50 PRINT#4,L$
60 PRINT L$
70 LOOP
80 CLOSE4:DCLOSE3:END
```

OUTPUT VERSO UN MODEM

Un modem è un'unità utilizzata per trasmettere e ricevere i dati attraverso le linee telefoniche.

Generalmente, un computer usa una porta RS232 per comunicare con il mondo esterno. La porta RS232C del Commodore (conosciuta anche come Porta dell'Utente) è una variante della porta standard RS232. L'RS232 ha valori di voltaggio per la TTL da 0 a 5, rispetto ai più/meno 12 volt standard.

La frase OPEN per il modem (unità 2) pone i parametri in modo che si combinino con la velocità (baud rate = bit al secondo) e con il formato (numero dei dati e bit di stop) del computer collegato inviando un codice di Registro di Controllo. Un secondo codice richiesto, il codice di Registro di Comando, definisce la parità, il duplex e l'handshaking.
Per esempio:

```
OPEN3,2,0,CHR$(6)CHR$(112)
```

apre un canale verso l'Ingresso dell'Utente (unità #2) e stabilisce un baud rate di 300, una parola di 8 bit, un singolo bit di stop (Registro di Controllo (6)), parità pari e metà duplex (Registro di Comando (112)). Osservate la Figura 13-1, Mappa dei Registri di Controllo e la Figura 13-2, Mappa dei Registri di Comando.

BIT 6	BIT 5	DATO LUNGH. PAROLA
0	0	8 BIT
0	1	7 BIT
1	0	6 BIT
1	1	5 BIT

3	2	1	0	BAUD RATE	
0	0	0	0	USER RATE	
0	0	0	1	50 BAUD	
0	0	1	0	75	
0	0	1	1	110	
0	1	0	0	134.5	
0	1	0	1	150	
0	1	1	0	300	
0	1	1	1	600	
1	0	0	0	1200	
1	0	0	1	1800	[NI]
1	0	1	0	2400	[NI]
1	0	1	1	3600	[NI]
1	1	0	0	4800	[NI]
1	1	0	1	7200	[NI]
1	1	1	0	9600	[NI]
1	1	1	1	19200	[NI]

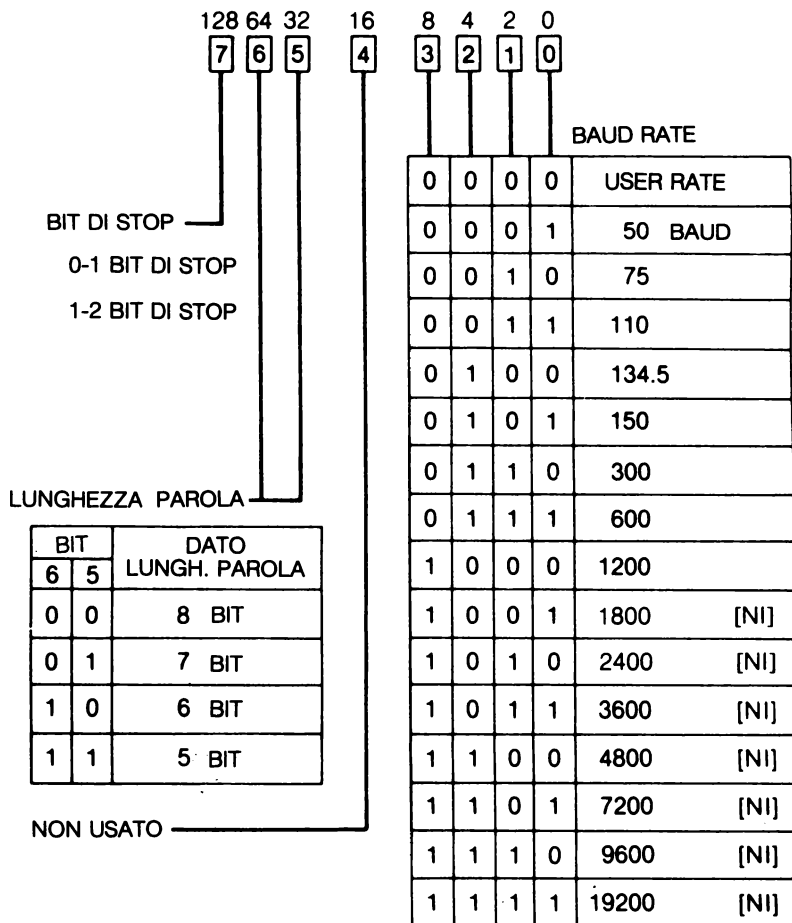



Figura 13-1. Mappa del Registro di Controllo

IL CANALE RS-232

Si può accedere alle capacità dell'RS-232 sia in BASIC che in linguaggio macchina. I comandi in BASIC **INPUT#** e **GET#** vengono usati per andare a prendere i dati dal buffer di ricezione mentre **PRINT#** e **CMD** pongono i dati nel buffer di trasmissione.

Le routine Kernal dell'RS-232 operano sotto il controllo dei timer e delle interruzioni dell'unità 6526 CIA #2. Il chip 6526 genera richieste NMI (Non-Maskable Interrupt = Interruzione Non Mascherabile) per il trattamento dell'RS-

BIT 7	BIT 6	BIT 5	OPERAZIONI
-	-	0	PARITÀ DISABILITATA, NESSUNO GENERATO/RICEV.
0	0	1	PARITÀ DISPARI RICEVITORE/TRASMETTITORE
0	1	1	PARITÀ PARI RICEVITORE/TRASMETTITORE
1	0	1	INDICATORE TRASMESSO CONTROLLO DI PARITÀ DISAB.
1	1	1	SPAZIO TRASMESSO CONTROLLO DI PARITÀ DISAB.

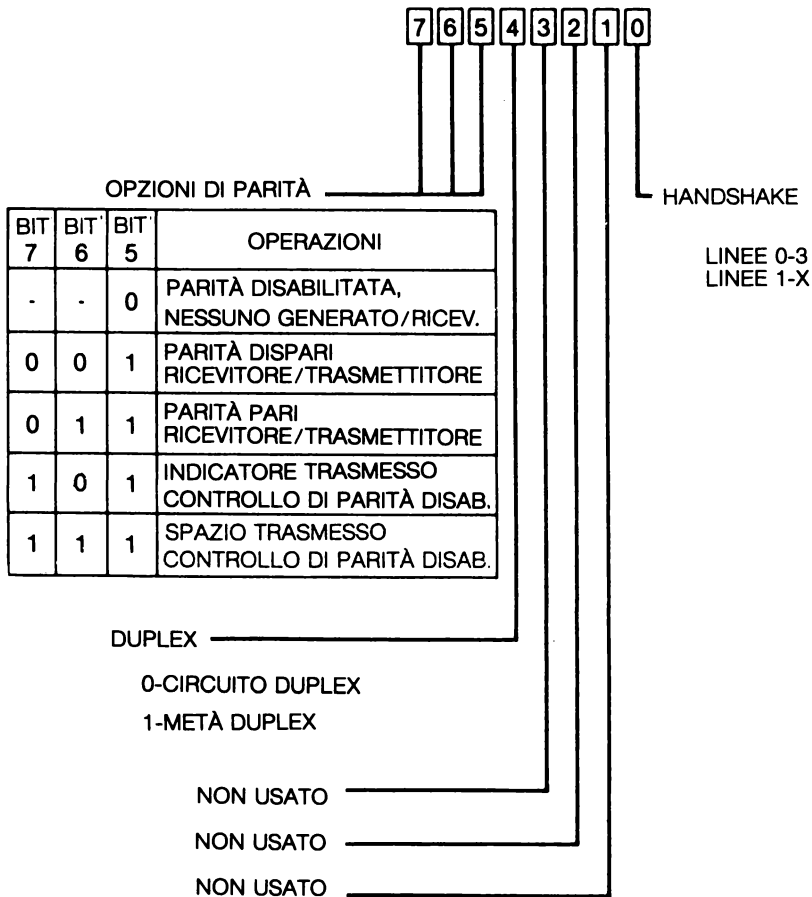



Figura 13-2. Mappa del Registro di Comando

232. Questo permette al trattamento di background dell'RS-232 di aver luogo nei programmi in BASIC ed in linguaggio macchina. Nel Kernal ci sono arresti incorporati, cassette e routine di bus seriali per prevenire la disgregazione della memoria dei dati o della trasmissione con i NMI che sono generati dalle routine dell'RS-232. Durante le attività delle cassette o dei bus seriali, i dati non possono essere ricevuti dalle unità dell'RS-232. Poichè questi arresti sono solo locali, (assumendo che voi siate attenti mentre programmate) non dovrebbero risultare interferenze.

Ci sono due buffer nella memoria del Commodore 128 per aiutare la prevenzione della perdita dei dati quando si trasmettono o si ricevono informazioni dell'RS-232.

I buffer di I/O dell'RS-232 del Commodore 128 consistono di due buffer first-in/first-out (FIFO), ognuno di 256 byte. In modo C128, i buffer di I/O dell'RS-232 sono allocati permanentemente. Il buffer di input è localizzato tra 3072 e 3327 ($0C00-$0CFF). Il buffer di output è localizzato tra 3328 e 3583 ($0D00-$0DFF). Non potete danneggiare la memorizzazione dei programmi aprendo e chiudendo il canale dell'RS-232 come può succedere per il Commodore 64. Allocando permanentemente i buffer in una locazione fissa, non può accadere un errore "OUT OF MEMORY" ed il vostro programma in memoria non sarà "sovrapposto".

APERTURA DEL CANALE RS-232

In ogni caso può essere aperto un solo canale RS-232; una seconda istruzione OPEN causa il reset dei puntatori del buffer. Qualsiasi carattere sia nel buffer di trasmissione che in quello di ricezione sarà perduto.

Nel campo del nome del file possono essere inviati fino a 4 caratteri. I primi due sono del registro di controllo e di comando; gli altri due sono usati per le esigenze dell'utente, e per la baud rate (bit al secondo) non standard. La baud rate, la parità ed altre opzioni possono essere selezionate con questa funzione.

Non c'è controllo di errore nella parola di controllo per riconoscere una baud rate non implementata. Qualunque parola di controllo illegale farà sì che l'output del sistema operi ad una velocità molto bassa (sotto 50 baud).

SINTASSI BASIC

OPEN lfn,2,0" <reg. di controllo> <reg. di comando> <opt baud bassa> <opt baud alta>"

Il numero del file logico (lfn) può essere qualsiasi numero tra 1 e 255. Ma siate consapevoli del fatto che se voi scegliete un numero di file logico maggiore di 127, allora una line feed seguirà tutti i ritorni del carrello (carriage returns).

IL REGISTRO DI CONTROLLO

Il Registro di Controllo è un singolo byte che viene richiesto per specificare le baud rate ed altre opzioni dell'RS-232. Selezionate i parametri che volete dalla Mappa dei Registri di Controllo (Figura 13-1) ponendo il valore appropriato del bit in forma binaria. Una selezione, per esempio, da sinistra a destra di due bit di stop, parola di 7 bit e baud rate 1200 dà:
1010 1000 (in binario), che in decimali è 168 ($A8) e risulta in

```
OPEN 3,2,0,CHR$(168)
```

definendo questi parametri.

IL REGISTRO DI COMANDO

Il **Registro di Comando** è un singolo byte che definisce parità, duplex e handshaking. Selezionate i parametri dalla Mappa dei Registri di Comando nella Figura 13-2 creando il codice binario e convertendolo in un valore decimale. Così:

OPEN 3,2,0,CHR$(168)CHR$(32)

specifica la parità dispari, il duplex pieno. Una baud rate diversa da quelle listate nella Mappa dei Registri di Controllo è selezionabile anche scegliendo 0000 per i bit da 0 a 3 nel Registro di Controllo ed includendo questa velocità nella istruzione OPEN come due byte che seguono il Registro di Comando. I valori di "opt baud bassa" e del byte alto sono calcolati dalla baud rate desiderata come una funzione della frequenza del sistema (1.02272710^6 per il sistema USA NTSC o 0.98524810^6 sistema PAL europeo). Dividete la frequenza per la velocità desiderata. Dividete questo a metà e sottraete 100. Questo rappresenta un numero decimale di due byte. Il valore alto è il numero di volte che esso può essere diviso per 256. Il resto è il byte basso.

ESEMPIO:

1000 Baud: Dividete la frequenza del sistema per la baud rate:

1022727/1000 = 1022.727

Dividete per 2 e sottraete 100:

1022.727/2−100 = 411.3635 (o arrotondato a 412)

Convertitelo in due valori di byte: la divisione per 256 dà 1 con un resto di 156. L'istruzione seguente perciò definisce una baud 1000, 7 bit, un singolo bit di stop, parità dispari, duplex pieno:

OPEN 3,2,0,CHR$(168)CHR$(32)CHR$(156)CHR$(1)

dove il valore del Registro di Controllo in numero binario è 00100000 ed il Registro di Comando in numero binario è 00100000.

NOTA: In un programma in BASIC del C64, il comando OPEN RS-232 dovrebbe essere dato prima di creare qualsiasi variabile o matrice perchè avviene un CLR automatico quando viene aperto un canale RS-232 facendo riferimento alla locazione dei 512 byte in cima alla memoria. Non è il caso del modo C128, poichè i buffer di I/O dell'RS-232 sono allocati permanentemente.

DATI DAL CANALE RS-232

Il buffer di ricezione del Commodore 128 ($0C00-$0CFF) terrà fino a 255 caratteri prima di riempirsi completamente. Questo viene indicato nella parola di stato dell'RS-232 **ST**. Se capita un riempimento, vengono persi i caratteri successivi. Tenete il buffer più pulito possibile. Per ricevere i dati ad alta velocità, usate le routine del linguaggio macchina o usate il comando FAST per procedere a 2MHz con lo schermo ad 80 colonne (solo in modo C128).
Usate GET# con il numero di file corrispondente alla istruzione OPEN. Le entrate Kernal in codice macchina includono BASIN e GETIN.

INVIO DEI DATI AL CANALE RS-232

Quando inviate un dato, il buffer di output ($0D00-$0DFF) può tenere 255 caratteri prima che avvenga un riempimento del buffer. Il sistema aspetterà nella routine CHROUT finchè non verrà permessa la trasmissione o i tasti **RUN/STOP** e **RESTORE** verranno usati per riprendere il sistema.

Usate CMD o PRINT# con il numero di file appropriato per inviare i dati. Le entrate Kernal accettabili includono BSOUT (CHROUT).

CHIUSURA DEL CANALE RS-232

Usate CLOSE seguito dal numero di file per chiudere il canale RS-232. La chiusura del file elimina tutti i dati nei buffer al momento dell'esecuzione, ferma tutto ciò che viene trasmesso e ricevuto e pone alte le linee dei dati RTS e quelle trasmesse. In modo C64 (solo) CLOSE rimuove entrambi i buffer.

NOTA: Dovete fare attenzione per assicurarvi che tutti i dati vengano trasmessi prima di chiudere il canale. Un modo per controllare questo dal BASIC è:

```
10 DO
20 LOOP WHILE PEEK (2575) e 3
30 CLOSE lfn
```

La locazione 2575 è la locazione "ENABL" dell'RS-232.

La tabella 13-1 definisce le linee del canale dell'Ingresso dell'Utente dell'RS-232.

(6526 UNITÀ #2 Loc.$DD00—$DD0F)						
IDENTIFIC. PIN	**6526**	**DESCRIZIONE**	**EIA**	**ABV**	**IN/OUT**	**MODI**
C	PB0	Dato Ricevuto	(BB)	Sin	IN	1 2
D	PB1	Richiesta da inviare	(CA)	RTS	OUT	1*2
E	PB2	Terminale del dato pronto	(CD)	DTR	OUT	1*2
F	PB3	Indicatore ad anello	(CE)	RI	IN	3
H	PB4	Segnale di linea ricevuta	(CF)	DCD	IN	2
J	PB5	Non assegnato	( )	XXX	IN	3
K	PB6	Sblocco per l'invio	(CB)	CTS	IN	2
L	PB7	Insieme del dato pronto	(CC)	DSR	IN	2
B	FLAG2	Dato ricevuto	(BB)	S_{in}	IN	1 2
M	PA2	Dato trasmesso	(BA)	S_{out}	OUT	1 2
A	GND	Massa protettiva	(AA)	GND		1 2
N	GND	Segnale di massa	(AB)	GND		1 2 3

MODI:
1. Interfaccia a 3 linee (S_{in}, S_{out}, GND).
2. Interfaccia a X linee.
3. Solo Disponibile per l'Utente (Non usato/non implementato nel codice).

* Queste linee sono tenute alte durante il modo a 3 linee.

Tabella 13-1. Linee dell'Ingresso dell'Utente

```
[7] [6] [5] [4] [3] [2] [1] [0] (Ling. Macchina —RSSTAT)
 :   :   :   :   :   :   :   :__BIT DI ERRORE DI PARITÀ
 :   :   :   :   :   :   :______BIT DI ERRORE DELLA FRAME
 :   :   :   :   :   :__________BIT DI OVERRUN DEL BUFFER RICEV.
 :   :   :   ·   :______________BUFFER RICEVENTE— VUOTO
                                (PER IL CONTROLLO DOPO UN GET#)
 :   :   :   :__________________BIT DI MANCATO SEGNALE CTS
 :   :   :______________________BIT NON USATO
 :   :__________________________BIT DI MANCATO SEGNALE DSR
 :______________________________BIT DI ROTTURA RICONOSCIUTA
```

Figura 13-3. Registro di stato RS-232

NOTE alla Figura 13-3:
Il Registro di Stato dell'RS-232 può essere letto con queste istruzioni. Se lo STATO = 0, allora non è stato trovato alcun errore.

S = PEEK (2580):PRINT S: POKE 2580,0

Se lo stato dell'RS-232 viene letto dal BASIC con le istruzioni precedenti o usando la routine KERNAL READST, la parola di stato dell'RS-232 viene cancellata quando uscite. Se sono necessari usi multipli della parola di STATO la variabile S qui sopra deve essere assegnata ad un'altra variabile. Per esempio:

SR = S:REM ASSEGNA S AD SR

Lo stato dell'RS-232 viene letto (e cancellato) solo quando il canale dell'RS-232 è stato l'ultimo I/O esterno usato.

CAMPIONE DI PROGRAMMA IN BASIC USANDO IL CANALE RS-232

Questo semplice programma al terminale funziona sia in modo C128 che in modo C64:

```
10 REM ESEMPIO DI EMULATORE DI TERMINALE STUPIDO
20 REM QUESTO PROGRAMMA NECESSITA L'USO DI UN MODEM
30 OPEN5,2,3,CHR$(6)+CHR$(0)
40 DIMF%(255),T%(355)
50 FORJ=32TO64:T%(J)=J:NEXT
60 T%(13)=13:T%(20)=8:RV=18:CT=0
70 FORJ=65TO90:K=J+32:T%(J)=K:NEXT
80 FORJ=91TO95:T%(J)=J:NEXT
90 FORJ=193TO218:K=J-128:T%(J)=K:NEXT
100 T%(146)=16:T%(133)=16
110 T%(137)=3:T%(134)=17:T%(138)=19
120 FORJ=0TO255
130 K=T%(J)
140 IFK<>0THENF%(K)=J:F%(K+128)=J
150 NEXT
160 PRINT" "CHR$(147)
170 GET#5,A$
180 IFA$=""ORST<>0THEN220
190 PRINT" "CHR$(157);CHR$(F%(ASC(A$)));
200 IFF%(ASC(A$))=34THENPOKE212,0
210 GOTO170
220 PRINTCHR$(RV)" "CHR$(157);CHR$(146);:GETA$
230 IFA$<>""THENPRINT#5,CHR$(T%(ASC(A$)));
240 CT=CT+1
250 IFCT=8THENCT=0:RV=164-RV
260 GOTO170
```

LOCAZIONI DI MEMORIA IN PAGINA ZERO E USO DELL'INTERFACCIA DEL SISTEMA PER L'RS-232

Le locazioni di memoria in pagina zero dell'RS-232 per entrambi i modi C128 e C64 sono i seguenti:
$00A7-INBIT--Ricevitore della memorizzazione temporanea del bit di input.
$00A8-BITCI--Ricevitore del bit di inclusione.
$00A9-RINONE--Ricevitore del controllo del bit di inizio lampeggio.
$00AA-RIDATA--Ricevitore della locazione del byte buffer/assemblaggio.
$00AB-RIPRTY--Ricevitore della memorizzazione del bit di parità.
$00B4-BITTS--Trasmettitore del bit di esclusione.
$00B5-NXTBIT--Trasmettitore del bit successivo che deve essere inviato.
$00B6-RODATA--Trasmettitore della locazione del byte buffer/disassemblaggio.

Tutte le locazioni in pagina zero qui sopra vengono usate localmente e vengono date solo come guida per la comprensione delle routine associate. Queste non possono essere usate direttamente dal programmatore del BASIC o del livello Kernal per eseguire funzioni di programmazione del tipo RS-232. Devono essere usate le routine del sistema RS-232.

LOCAZIONI DI MEMORIA DI PAGINA NON ZERO ED USO DELL'INTERFACCIA DEL SISTEMA PER L'RS-232

Le locazioni generali di memoria (di pagina non zero) dell'RS-232 per il modo C128 sono le seguenti: (gli indirizzi tra parentesi sono per il modo C64.)

$0A10($0293)-M51CTR--Pseudo Registro di Controllo del 6551 (Figura 13-1).
$0A11($0294)-M51CTR--Pseudo Registro di Comando del 6551 (Figura 13-2).
$A012($0295)-M51AJB--Due byte che seguono i Registri di Controllo e di Comando nel campo del nome del file. Queste locazioni contengono la baud rate definita dall'utente per l'inizio del test del bit durante l'attività dell'interfaccia che, a turno, viene usato per calcolare la baud rate.
$0A14($0297)-RSSTAT--Il Registro di Stato dell'RS-232 (Figura 13-3).
$0A15($0298)-BITNUM--Il numero dei bit che devono essere inviati/ricevuti.
$0A16($0299)-BAUDOF--Due byte che sono uguali al tempo di una cella del bit (basato sul clock del sistema/baud rate).
$0A18($029B)-RIDBE--L'indice dei byte alla fine del buffer di ricezione FIFO.
$0A19($029C)-RIDBS--L'indice dei byte all'inizio del buffer di ricezione FIFO.
$0A1A($029D)-RODBS--L'indice dei byte all'inizio del buffer di trasmissione FIFO.
$0A1B($029E)-RODBE--L'indice dei byte alla fine del buffer di trasmissione FIFO.
$0A0F($02A1)-ENABL--Tiene le interruzioni correnti attive in CIA#2 ICR. Quando il bit 4 viene acceso, il sistema aspetta il segnale del Ricevitore. Quando il bit 1 è acceso, allora il sistema riceve i dati. Quando è acceso il bit 0, il sistema trasmette i dati.

OUTPUT VERSO UNO SCHERMO

La forma più comune di output di un computer (chiamata soft copy) è un tubo di visualizzazione che visualizza dei punti luminescenti sullo schermo controllati da segnali elettrici. Sebbene il televisore sia un monitor specializzato e limitato dalla densità dei punti prescritta dagli standard FCC per una ricezione adeguata, esso è disponibile in molte case e diventa un'unità di output pronta per l'uso per il computer domestico. Il C128 emette segnali di frequenza radio attraverso la porta della TV verso l'antenna ed il sintonizzatore del televisore.

I monitor dei computer producono video di qualità migliore poichè sono cablati direttamente e non hanno bisogno di sintonizzatori. Il C128 include una porta che accetta i monitor tradizionali e il monitor a colori 1702 a 40 colonne del Commodore. La capacità ad 80 colonne a colori del C128 viene aumentata dal nuovo Modello 1902 di monitor a colori a 40/80 colonne. In questo monitor RGBI, i segnali controllano separatamente i tre colori (rosso, verde, blu) per ottenere il massimo della chiarezza e dei dettagli. Si accede allo schermo come ad un'unità di input o di output con il comando OPEN dispositivo 3.

```
OPEN1,3
```

Lo schermo è l'unità di output per default del C128. Digitando sulla tastiera (unità 0) i caratteri risultano in un output di visualizzazione dello schermo tutte le volte che viene inserito il monitor. Normalmente non sono richiesti comandi speciali. Tuttavia, una volta che il controllo viene mandato ad un'altra unità, si può richiedere l'inversione dello schermo.

Il controllo dello schermo si ottiene principalmente con il comando PRINT; la punteggiatura associata include le virgolette, punto e virgola e le comuni funzioni in BASIC comprese TAB(X), SPC(X) e CHR$(X) ed il comando POKE. Inoltre, ci sono i controlli del colore, animazione degli sprite e tasti grafici.

Se usate il monitor duale 1902 o due monitor separati da 40 ed 80 colonne, è possibile scambiare il controllo tra i due schermi con il commutatore **40/80** o premendo i tasti **ESC** e X. Se uno è usato per il testo e l'altro per la grafica, usate il comando GRAPHIC.

Il comando seguente scambierà anche l'output video dell'unità:

```
PRINT CHR$(27)"X"
```

In aggiunta ad uno schermo, possono essere usati dei comandi per il suono attraverso un monitor Commodore o gli altoparlanti della televisione. Consultate il Capitolo 12, il Suono e la Musica.

OUTPUT VERSO IL DATASSETTE

I programmi ed i file del Commodore possono essere memorizzati anche su una cassetta da registratore usando un registratore speciale chiamato *Datassette.* Diversamente dalla maggior parte dei registratori, che memorizzano segnali sinusoidali analoghi, il Datassette memorizza segnali ad impulsi digitali che sono più distinti, risultando in un formato di memoria affidabile. Confrontato con un sistema di memorizzazione dei dischetti, il registratore è un pò più lento e limitato con i file di programma e sequenziali. L'accesso ai file del registratore è sequenziale.

Come **DLOAD** carica automaticamente un programma dal dischetto, **LOAD** carica un programma dal registratore. In modo simile **SAVE** e **VERIFY** portano automaticamente al registratore. Un LOAD per il codice macchina che richiede un indirizzo secondario sarebbe:

```
LOAD"NOME",1,1
```

L'accesso al registratore per i file sequenziali viene ottenuto con un'istruzione OPEN verso l'unità 1:

```
OPEN1,1
```

Un indirizzo secondario 0 (valore di default) assume che il file sia aperto per la lettura (usando INPUT# o GET#). Un indirizzo secondario per la scrittura (istruzioni PRINT#) richiede un 1 o 2, dove 2 include un marcatore di fine nastro.

```
OPEN1,1,2"NOME DEL FILE"
```

Il marcatore di fine nastro fa sì che non si cerchino i file oltre il file dei dati che state cercando. Viene scritto quando viene dato il comando CLOSE.

NOTA: Non collegate il Datassette ed il drive 1571 al C128 contemporaneamente. Se li usate allo stesso tempo, i trasferimenti dei dati possono non essere affidabili.

INPUT DALLE PORTE DI CONTROLLO

Il C128 ha due porte di controllo (numerate 1 e 2) che permettono l'uso di dispositivi di controllo come mouse, joystick, paddle o penna ottica. Ogni porta dei giochi accetterà o un joystick, un mouse o una coppia di paddle. Una penna ottica può essere inserita solo nella Porta 1.

I JOYSTICK

Un joystick digitale ha cinque interruttori. Quattro sono usati per la direzione ed uno per sparare. Sono sistemati come in Figura 13-4.

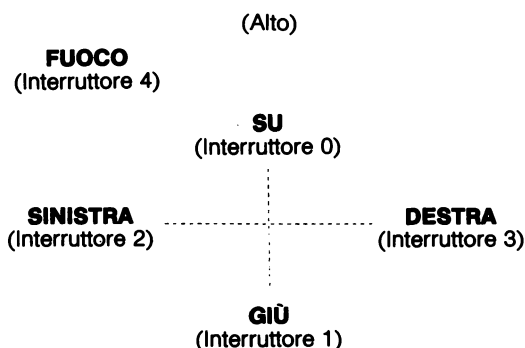



Figura 13-4. Disposizione degli Interruttori del Joystick

Questi interruttori corrispondono ai cinque bit più bassi dei dati alla locazione 56320 o 56321. Normalmente il bit è posto a 1 se una direzione non viene scelta o il tasto per sparare non viene premuto. Se il tasto per sparare viene premuto, il bit (il bit 4 in questo caso) diventa 0. Per leggere il joystick dal BASIC del C64, deve essere usata la seguente subroutine:

```
10 REM IL JOYSTICK LEGGE ROUTINE PER LE MODALITA' C128 E C64
20 JV=PEEK(56321)
30 IF (JV AND 15)<15 THEN PRINT 15-(JV AND 15)
40 FIRE=JV AND 16
50 IF FIRE<> 16 THEN PRINT"FUOCO"
60 GOTO 20
```

NOTA: Per il primo joystick, ponete JV = PEEK (56320).

I valori per JV corrispondono alle direzioni del joystick mostrate nella Tabella 13-2.

JV UGUALE A	DIREZIONE
0	NESSUNA
1	SU
2	GIÙ
3	— —
4	SINISTRA
5	SU E SINISTRA
6	GIÙ E SINISTRA
7	— —
8	DESTRA
9	SU E DESTRA
10	GIÙ E DESTRA

Tabella 13-2. Valori di direzione del Joystick

Una routine in linguaggio macchina che esegua lo stesso compito viene data nella sezione sulle istruzioni shift e rotate nel Capitolo 6, Il Linguaggio Macchina sul Commodore 128.

Il BASIC 7.0 include speciali funzioni per il joystick. Per determinare la posizione del joystick N, **PRINT JOY(N)** rimanda alla posizione del joystick. Un numero tra 0 e 8 incluso specifica la posizione relativa dove 0 è stazionaria e da 1 a 8 rappresentano le posizioni in senso orario. I valori tra 128 e 136 dicono che il tasto per sparare è premuto. Così se **JOY(2)** rimanda a 131, significa che il joystick numero 2 spara a destra (posizione 3).

```
10 REM ROUTINE PER IL JOYSTICK IN MODALITA' C128 PER MUOVERE GLI SPRITE
20 SPRITE 1,1,7:REM ABILITA LO SPRITE 1
30 MOVSPR 1,160,150:REM POSIZIONA LO SPRITE 1 AL CENTRO DELLO SCHERMO
40 DO
50 :  DO
60 :     A=JOY(1):REM ASSEGNA AL JOYSTICK IL VALORE DELLA DIREZIONE
70 :     IF A=0 THEN LOOP:REM CICLO SE ZERO - CIO' MANTIENE LO SPRITE STAZIONARIO
80 :     B=A:REM PONE B UGUALE AD A
90 :     Z=(A-1)*45:REM CALCOLO DELL'ANGOLO
100 :    MOVSPR 1,5;Z:REM MUOVE LO SPRITE DI 5 UNITA' CON IL CALCOLO DELL'ANGOLO
110 :    A=JOY(1):REM TESTA IL PROSSIMO VALORE DEL JOYSTICK
120 : LOOP WHILE A=B:REM MUOVE NELLA STESSA DIREZIONE FINCHE' IL NUOVO VALORE E'
      UGUALE AL VECCHIO VALORE DEL JOYSTICK
130 LOOP:REM ALTRIMENTI TESTA UN ALTRO  VALORE DEL JOYSTICK E MUOVE IN UNA DIRE-
    ZIONE DIFFERENTE
```

Usando la logica booleana e la tecnica di mascheramento, si può fare un confronto per determinare se il tasto per sparare era stato premuto:

IF (JOY(N)AND 128) = 128 THEN PRINT "FUOCO"

IL MOUSE 1350

Il Commodore 128 fa uso di una nuova unità periferica in più conosciuta come mouse. ·Il mouse Commodore è un'interfaccia dell'utente che vi permette di controllare le operazioni di un programma, come la selezione di un menu o di opzioni di icone, senza la necessità di digitare sulla tastiera. Si collega alle porte dell unità di controllo come un joystick, un paddle o una penna ottica. I programmi che sopportano un mouse di solito hanno un indicatore/puntatore che vi fa capire dove il mouse sta puntando sullo schermo. Per muovere il puntatore sullo schermo, ponete il mouse su una superficie piana e muovetelo nella direzione in cui volete muovere il puntatore. Alzate il mouse dalla superficie e l'indicatore dello schermo rimane fermo. Il tasto a sinistra risponde come un tasto per sparare del joystick. Il tasto a destra può avere delle funzioni in più come viene specificato da un particolare pacchetto di software.

Per quanto concerne il BASIC, il mouse è sostenuto dalla funzione JOY, nello stesso modo in cui viene controllato un joystick. L'esempio precedente di programma del joystick, che muove uno sprite sullo schermo, funziona anche con il mouse. Anche il programma Musicista con il Joystick nel Capitolo 12 funziona con il mouse. Qualsiasi programma in BASIC che usi la funzione per il joystick JOY dovrebbe operare con il mouse del Commodore.

Il mouse funziona anche in modo C64 nello stesso modo in cui una routine joystick funzionerebbe in modo C64. L'esempio precedente di programma del joystick in modo C64 sostiene il mouse in modo C64.

I PADDLE

Un paddle è connesso sia al chip CIA-1 che al SID (MOS 6581 Sound Interface Device = Unità di Collegamento Sonoro) attraverso una porta di controllo. Il valore del paddle viene letto con i registri SID 54297 ($D419) e 54298 ($D41A). *I paddle non sono affidabili se letti solo dal BASIC*! Il modo migliore per usare i paddle, con il BASIC o un codice macchina è di usare una routine in linguaggio macchina (fate il SYS verso essa dal BASIC poi fate il PEEK delle locazioni di memoria usate dalla subroutine).

Il programma seguente in BASIC muove uno sprite con i paddle:

```
10 REM ROUTINE IN MODALITA' C128 PER MUOVERE GLI SPRITE CON MANOPOLE
20 REM POSIZIONA LE DUE MANOPOLE NELLA PORTA A
30 REM LA MANOPOLA 1 CONTROLLA IL MOVIMENTO DEGLI SPRITE ORIZZONTALI
40 REM LA NONOPOLA 2 CONTROLLA QUELLO VERTICALE
50 SPRITE 1,1,7
60 DO
70 A=POT(1):REM ASSEGNA LA POSIZIONE (X) DELLA MANOPOLA 1
80 B=POT(2):REM ASSEGNA LA SECONDA POSIZONE
90 PRINT A;B:REM VISUALIZZA I VALORI
100 MOVSPR 1,A,B:REM MUOVE LO SPRITE 1 IN ACCORDO CON I VALORI DELLA MANOPOLA
110 IF (A>255) OR (B>255) THEN BEGIN:REM TESTA IL PULSANTE DEL FUOCO
120 :       SOUND 2,64000,60,0,1000, 16000,3:REM FUOCO
130 :       COLOR 0,3:REM CAMBIA IL COLORE DELLO SCHERMO
140 :       SLEEP 1  :REM RITARDO
150 :       COLOR 0,1:REM CAMBIA IL COLORE DELLO SFONDO
160 BEND
170 LOOP:REM RIPRESA DI UN ALTRO CICLO
```

PENNA OTTICA

L'input della penna ottica trattiene la posizione corrente dello schermo in un paio di registri (LPX, LPY) che commutano su un fronte di discesa del segnale. Il registro 53267 ($D013) della posizione X conterrà l'MSB 8 della posizione X al momento della transizione. Poichè la posizione X è definita da un contatore di stato 512 (9 bit), viene data una risoluzione di due punti orizzontali. In modo simile, la posizione Y è trattenuta nel suo registro 53268 ($D014), ma qui 8 bit danno una risoluzione di un singolo raster all'interno dello schermo visibile. Il latch (dispositivo in grado di trattenere un segnale) della penna ottica può essere attivato solo una volta per frame ed attivazioni successive all'interno della stessa frame non avranno effetto. Perciò dovete prendere parecchi campioni prima di portare la penna allo schermo (tre o più campioni come media), secondo le caratteristiche della vostra penna ottica. Il programma seguente è una routine di disegno con la penna ottica in modo C128:

```
10 REM ROUTINE DI DISEGNO IN MODALITA' C128 PER PENNA OTTICA
20 REM PORRE LA PENNA OTTICA NELLA PORTA DI CONTROLLO 1
30 COLOR 0,2:COLOR 4,2:COLOR 1,1:REM SELEZIONA IL COLORE DELLO SCHERMO
40 GRAPHIC 1,1:REM MODALITA' ALTA RISOLUZIONE
50 DO
60 :     X=PEN(0)     :REM ASSEGNA LE COORDINATE ORIZZONTALI
70 :     Y=PEN(1)     :REM ASSEGNA QUELLE VERTICALI
80 :     DRAW 1,X,Y :REM RILEVA IL PUNTO
90 LOOP :REM RIPRESA DEL CICLO
100 :
110 REM SE TI PIACE IL DISEGNO,
120 REM FERMA IL PROGRAMMA E
130 REM SALVA QUESTO COME UN FILE BINARIO
140 REM A PARTIRE DA $2000-$3FFF.
```

CONTROLLO OUTPUT VERSO TUTTE LE UNITÀ

Ogni unità periferica è identificata da un numero di unità, come in Tabella 13-3.

UNITÀ	NUMERO DI UNITÀ	INDIRIZZI SECONDARI
Tastiera	**0**	
Datasette	**1**	**0,1,2**
Modem (RS-232)	**2**	**0**
Schermo	**3**	**0,1**
Stampante	**4 o 5**	**0,7,ecc**
Disk Drive	**8,9,10,11**	**2-14,15**

Tabella 13-3. Numeri di dispositivo

Le unità seriali possono essere numerate da 4 a 30, ma di solito esse sono conformi alle convenzioni di cui sopra. Si può accedere a tutte le unità con una singola frase OPEN usando una variabile intera:

```
10 INPUT "ASSEGNA IL NUMERO DI SISTEMA";D%
20 INPUT "ASSEGNA L'INDIRIZZO SECONDARIO";S%
30 INPUT "NOME DEL FILE O DEL REGISTRQ DI CONTROLLO";A$
40 OPEN 9,D%,S%,A$
50 GOSUB 80
60 CLOSE 9
70 END
80 REM PONI QUI LA SUBROUTINE
90 RETURN
```

PINOUT DI INPUT/OUTPUT

LA PORTA SERIALE

La Porta Seriale è una disposizione concatenata a margherita progettata per permettere al Commodore 128 di comunicare con più di un'unità, come i drive e le stampanti Commodore. Si possono connettere fino a cinque unità contemporaneamente. Il bus seriale può trasmettere segnali di controllo dal computer, inviare i dati nel bus e ricevere i dati dal bus. I dati ed i segnali di controllo vengono mandati alla unità appropriata aprendo un indirizzo specifico del bus che va da 4 a 31. Di solito gli indirizzi 4 e 5 sono riservati alle stampanti, e gli indirizzi da 8 a 11 sono per il drive.

Nell'operazione del bus seriale vengono usate sei linee, tre input e tre output. Le tre linee di input portano i dati, i segnali di controllo e di temporizzazione nel computer. Le tre linee di output inviano i dati, i segnali di controllo e di temporizzazione dal computer alle unità esterne. Consultate le Specificazioni Hardware (Vol. 7, Capitolo 17) per i dettagli sul protocollo del bus seriale del C128.

I/O SERIALE

PIN	TIPO
1	SRQIN SERIALE
2	GND
3	ATN IN/OUT SERIALE
4	CLK IN/OUT SERIALE
5	DATI IN/OUT SERIALE
6	RESET

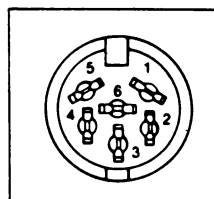



Figura 13-5. Pinout (disposizione dei piedini nelle porte di collegamento) della Porta Seriale

INGRESSO UTENTE (CANALE RS-232)

L'Ingresso dell'Utente (Canale RS-232) connette il Commodore 128 al mondo esterno, come fosse un altro computer Commodore. Esso è connesso direttamente al chip CIA 6526 #1 (di cui si tratta anche nel capitolo sull'hardware). La Figura 13-6 mostra i pinout dell'Ingresso dell'Utente, la Tabella 13-4 li definisce.

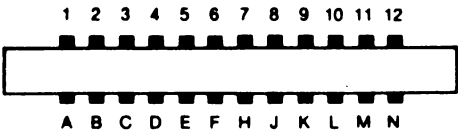



Figura 13-6. Pinout dell'Ingresso dell'Utente

PIN LATO ALTO	DESCRIZIONE	NOTE
1	GROUND	
2	+5V	(100 mA MAX.)
3	RESET	Collegando a massa questo pin, il Commodore 128 farà una partenza fredda, resettando completamente. I puntatori di un programma in BASIC saranno resettati, ma la memoria non sarà cancellata. Questo è anche un output RESET per le unità esterne.
4	CNT1	Contatore della porta seriale da CIA-1 (vedere specifiche CIA).
5	SP1	Porta seriale da CIA-1 (vedere specifiche CIA 6526).
6	CNT2	Contatore della porta seriale da CIA-2 (vedere specifiche CIA).
7	SP2	Porta seriale da CIA-1 (vedere specifiche CIA 6526).
8	PC2	Linea di handshaking da CIA-2 (vedere specifiche CIA).
9	SERIAL ATN IN	Questo pin è connesso alla linea ATN del bus seriale.
10	9 VAC + fase	Connesso direttamente al trasformatore del Commodore 128 (100 mA MAX.).
11	9 VAC - fase	
12	GROUND	
LATO IN BASSO		
A	GROUND	
B	FLAG2	Il Commodore 128 vi dà un controllo della Porta B sul chip 1 CIA. Sono disponibili otto linee per l'input o per l'output, come pure due linee per l'handshaking con un'unità esterna. Le linee di I/O per la Porta B sono controllate da due locazioni. Una è la Porta stessa ed è localizzata in 56577 ($DD01 esadec.). Naturalmente voi fate il PEEK per leggere un INPUT o il POKE per porre un OUTPUT. Ognuna delle otto linee di I/O può essere predisposta come INPUT o OUTPUT ponendo in modo appropriato il Registro di Direzione dei Dati.
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	GROUND	

Tabella 13-4. Descrizione dei pin dell'Ingresso dell'Utente

Il **Registro di Direzione dei Dati** ha la locazione in 56579 ($DD03 esadecimale). Ognuna delle otto linee nella porta ha un bit nel Registro di Direzione dei Dati a 8 bit (DDR) che controlla se quella linea sarà di input o di output. Se un bit nel DDR è un 1, la linea corrispondente della porta sarà un output. Se un bit nel DDR è uno 0, la linea corrispondente della porta sarà un input. Se il bit 3 del DDR è posto a 1, la linea 3 della porta sarà un output. Se il DDR è posto così:

BIT#	:	7 6 5 4 3 2 1 0
VALORE	:	0 0 1 1 1 0 0 0

le linee 5, 4 e 3 saranno output poichè quei bit sono 1. Le altre linee saranno input, poichè quei bit sono 0.

Per eseguire il PEEK o il POKE dell'Ingresso dell'Utente è necessario usare sia il DDR che l'ingresso stesso. Ricordate che le frasi PEEK e POKE necessitano di un numero da 0 a 255. I numeri dati nell'esempio devono essere tradotti in decimali prima di poter essere usati. Il valore sarebbe:

$2^5 + 2^4 + 2^3 = 32 + 16 + 8 = 56$

Notate che il numero del bit per il DDR è lo stesso numero che è uguale a 2 elevato ad una potenza che accenda il valore del bit.

(16 = 2 ↑ 4 = 2 per 2 per 2 per 2, 8 = 2 ↑ 3 = 2 per 2 per 2)

Le altre due linee, flag 2 e PA2, sono diverse dal resto dell'Ingresso dell'Utente. Queste due linee sono per lo più per l'handshaking e sono programmate diversamente dall'ingresso B.

Si ha bisogno dell'handshaking quando due unità comunicano. Poichè un'unità può girare ad una velocità diversa rispetto ad un'altra unità, è necessario dare ad ogni unità un modo di sapere che cosa stia facendo l'altra unità. Anche quando le unità stanno operando alla stessa velocità, è necessario l'handshaking per comunicare quando il dato deve essere inviato, e se è stato ricevuto. La linea del flag 2 ha caratteristiche adatte per l'handshaking.

Il flag 2 è un input che commuta sul fronte negativo del segnale (negative-edge-sensitive input) che può essere usato come input generalizzato di interruzione. Qualsiasi transizione negativa sulla linea del flag porrà il bit di interruzione del flag. Se l'interruzione del flag è inserita, questo causerà una **Richiesta di Interruzione**. Se il bit del flag non è abilitato, può essere interrogato/chiamato dal Registro di Interruzione durante il controllo del programma.

PA2 è il bit 2 della porta A del CIA. Viene controllato come qualsiasi altro bit della porta. La porta è localizzata in 56576 ($DD00). Il Registro della Direzione dei Dati è localizzato in 56578 ($DD02).

IL CONNETTORE VIDEO COMPOSITO

Questo connettore DIN fornisce i segnali di audio diretto e di video composito. Questi possono essere connessi al monitor del Commodore o usati con com-

ponenti separati. Questo è il connettore output a 40 colonne. La Figura 13-7 mostra i pinout (le disposizioni dei piedini del connettore) per il connettore del video composito. La Tabella 13-5 descrive i pinout.

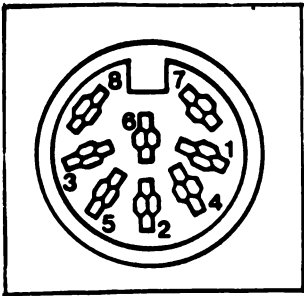



Figura 13-7. Pinout del Connettore del Video composto

PIN	TIPO	NOTA
1	LUM/SYNC	Luminosità/output SYNC
2	GND	
3	AUDIO OUT	
4	VIDEO OUT	Output del segnale composto
5	AUDIO IN	
6	COLORE OUT	Output del segnale colore
7	NC	Nessuna connessione
8	NC	Nessuna connessione

Tabella 13-5. Descrizioni dei pin del Connettore del Video composto

IL CONNETTORE VIDEO RGBI

Il connettore video RGBI è un connettore a 9 piedini che fornisce un segnale RGBI (Red(Rosso)/Green(Verde)/Blue(Blu)/Intensity(Intensità)). Questo è l'output ad 80 colonne. La Figura 13-8 mostra i pinout RGBI. La Tabella 13-6 definisce i pinout RGBI.

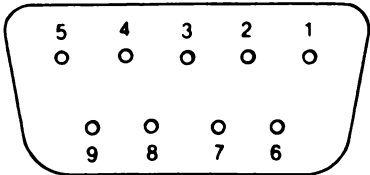



Figura 13-8. Pinout del Connettore RGBI

PIN	SEGNALE
1	Massa
2	Massa
3	Rosso
4	Verde
5	Blu
6	Intensità
7	Monocromatico
8	Sync Orizzontale
9	Sync Verticale

Tabella 13-6. Descrizione dei pin del Connettore RGBI

IL CONNETTORE PER IL REGISTRATORE

Un registratore Datassette 1530 può essere collegato alla porta della cassetta per memorizzare i programmi e le informazioni. La Figura 13-9 mostra i pinout della porta della cassetta. La Tabella 13-7 descrive i pinout.

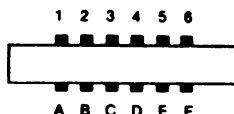



Figura 13-9. Pinout della Porta della cassetta

PIN	TIPO
A−1	GND
B−2	+5V
C−3	MOTORE DELLA CASSETTA
D−4	LETTURA DELLA CASSETTA
E−5	SCRITTURA DELLA CASSETTA
F−6	CONTROLLO DELLA CASSETTA

Tabella 13-7. Descrizioni dei Pin della Porta della cassetta

LE PORTE DI CONTROLLO

Ci sono due porte di controllo, che hanno i numeri 1 e 2. Ogni porta di controllo può accettare un joystick, un mouse o un paddle di controllo per i giochi. Una penna ottica può essere innestata solo nella Porta 1, la porta più vicina alla parte frontale del computer. Usate le porte come da istruzioni software. La Figura 13-10 mostra i pinout della Porta di Controllo. La Tabella 13-8 descrive i pinout.

1 2 3 4 5
6 7 8 9

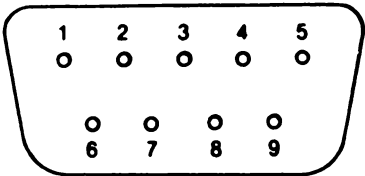



Figura 13-10. Pinout della Porta di controllo

PORTA DI CONTROLLO 1			PORTA DI CONTROLLO 2		
PIN	**TIPO**	**NOTA**	**PIN**	**TIPO**	**NOTA**
1	JOYA0		1	JOYB0	
2	JOYA1		2	JOYB1	
3	JOYA2		3	JOYB2	
4	JOYA3		4	JOYB3	
5	POT AY		5	POT BY	
6	BUTTON A/LP		6	BUTTON B	
7	+5V	MAX. 50mA	7	+5V	MAX. 50mA
8	GND		8	GND	
9	POT AX		9	POT BX	

Tabella 13-8. Descrizione dei pin della Porta di controllo

LA PORTA DI ESPANSIONE

Il connettore della porta di espansione è un connettore femmina (female edge) di 44 pin che può accedere ai bus indirizzo e dati del computer. Accetta le cartucce software Commodore preprogrammate di giochi e di utility. In qualità di porta in parallelo, può accettare i periferici IEEE Commodore con un'interfaccia IEEE per la strumentazione di controllo ed altre unità. I moduli di espansione RAM saranno connessi con questa porta ed accetteranno i comandi BASIC FETCH, STASH e SWAP.

La Figura 13-11 mostra i pinout della porta di espansione. La Tabella 12-9 descrive i pinout.

(visione della porta dal retro del C128)

Figura 13-11. Pinout della Porta di Espansione

PIN	TIPO	PIN	TIPO
1	GND	A	GND
2	+ 5V	B	ROMH
3	+ 5V	C	RESET
4	IRQ	D	NMI
5	R/W	E	S 02
6	Dot Clock	F	A15
7	I/O 1	H	A14
8	GAME	J	A13
9	EXROM	K	A12
10	I/O 2	L	A11
11	ROML	M	A10
12	BA	N	A9
13	DMA	P	A8
14	D7	R	A7
15	D6	S	A6
16	D5	T	A5
17	D4	U	A4
18	D3	V	A3
19	D2	W	A2
20	D1	X	A1
21	D0	Y	A0
22	GND	Z	GND

Tabella 13-9. Descrizione dei pin della Porta di Espansione

14

IL SISTEMA OPERATIVO DEL COMMODORE 128

Il sistema operativo del Commodore 128 controlla, direttamente o indirettamente, tutte le funzioni del vostro computer. Il sistema operativo del Commodore 128 è situato in un chip della ROM chiamato **Kernal,** che contiene circa 16K di istruzioni in linguaggio macchina. Queste istruzioni costituiscono le routine che controllano tutte le funzioni della macchina, anche quelle date per scontate. Per esempio, il Kernal controlla tutte le funzioni di input e di output, compresa la ricezione dei caratteri dalla tastiera quando li premete, l'invio del testo alla stampante, e la visualizzazione dei grafici e del testo sullo schermo. Ogni compito eseguito dal computer diverso dalle attività dei programmi di applicazione è controllato dal Kernal. Il Kernal gestisce ed esegue anche i programmi di applicazione che caricate o inserite da tastiera nella memoria del vostro computer.

COME SFRUTTARE COMPLETAMENTE IL SISTEMA OPERATIVO DEL COMMODORE 128

I 16K di istruzioni del sistema operativo contenuti nel Kernal sono disponibili per l'uso dei vostri programmi. Invece di "rinventare la ruota" ed il codice di duplicazione, potete chiamare (cioè usare) queste routine Kernal nei vostri programmi. Potete fare ciò con la Tabella di Salto Kernal (Kernal Jump Table), che consiste di una serie di punti di entrata della ROM nei quali potete chiamare le routine in linguaggio macchina già disponibili nel Kernal del Commodore 128. Chiamando queste routine, che gestiscono le funzioni fondamentali del computer, evitate il codice di duplicazione. Questo vi aiuta nell'utilizzo del vostro computer al massimo del suo potenziale.

La Tabella di Salto Kernal facilita anche la compatibilità. Se il sistema operativo del Commodore 128 è modificato o avanzato di grado, cosa che succede di frequente nell'industria dei microcomputer, i punti di entrata nella tabella di salto vengono revisionati perchè riflettano i cambiamenti di indirizzo delle subroutine Kernal. Il modo per mantenere compatibili i programmi di applicazione da una versione all'altra del sistema operativo è di entrare nel sistema operativo con la Tabella di Salto Kernal. Invece di saltare direttamente ad una subroutine (JSR), usate la Tabella di Salto Kernal come punto di entrata, poichè contiene il corretto vettore di indirizzo verso la routine specificata, e non tiene conto della versione del Kernal che viene usata. Se entrate sempre nelle routine del sistema operativo dalla Tabella di Salto Kernal, l'indirizzo della routine desiderata sarà sempre raggiunto. D'altra parte, se tralasciate la Tabella di Salto Kernal e saltate direttamente all'indirizzo dove pensate risieda la routine, potete causare un errore, poichè il punto di partenza della routine desiderata può essere cambiato da una versione del sistema operativo ad un'altra.

COME USARE (CHIAMARE) LE ROUTINE KERNAL NEI VOSTRI PROGRAMMI

La maggior parte delle routine del sistema richiedono dei parametri specifici, o valori, che devono essere caricati nell'accumulatore, nei registri di indice X o Y. Alcune subroutine Kernal richiedono che ulteriori routine di preparazione vengano chiamate prima di chiamare la routine specifica che si ha come obiettivo. Ogni subroutine Kernal termina con un'istruzione RTS che dice al microprocessore di ritornare dalla subroutine. Molte delle routine Kernal restituiscono dei valori che sono posti nell'accumulatore, nei registri X o Y. Alcune restituiscono anche codici di errore nel registro di stato, su cui potete agire nei vostri programmi di applicazione.

Ecco la procedura per chiamare una subroutine Kernal:

1. Ponete i valori preparatori necessari nei registri richiesti, sia A, X o Y.

2. Abilitate la configurazione appropriata del sistema. Per esempio, se la routine richiede la presenza del Kernal, chiamate le configurazioni 12, 13, 14 o 15, poichè queste rendono disponibile il Kernal. Il banco 15 è la configurazione per default.

3. Chiamate la subroutine con l'istruzione JSR, usando l'indirizzo del vettore di salto come mostrato nella Tabella di Salto Kernal nella Figura 14-1.

Per esempio, l'indirizzo di partenza della routine ACPTR è memorizzata, cominciando dalla locazione $FFA5. In realtà il codice operativo (opcode) per l'istruzione JMP è memorizzata alla locazione $FFA5 e l'indirizzo del punto di entrata della routine è memorizzata alla locazione $FFA6 e $FFA7. Ciò che in realtà succede è che l'istruzione JSR nel vostro programma di applicazione trasferisce il controllo all'indirizzo della tabella di salto ($FFA5, per esempio); poi l'istruzione JMP in $FFA5 salta alla subroutine nell'indirizzo specificato nelle locazioni $FFA6 e $FFA7. In altre parole, quando date un'istruzione JSR alla Tabella di Salto Kernal, in realtà eseguite due salti: uno (JSR) verso la tabella di salto, e poi un secondo salto (JMP) verso il reale indirizzo di partenza della routine.

La routine termina con un'istruzione RTS che è già parte della subroutine Kernal. Il vostro programma di applicazione riprende con l'istruzione immediatamente seguente l'istruzione JSR.

4. Durante il ritorno dalla subroutine, controllate il registro di stato per qualsiasi condizione di errore. Se è presente una condizione di errore, prendete delle

precauzioni nel vostro programma di applicazione per gestire l'errore ed agire sul valore del registro di stato. La Figura 14-1 è la tabella del vettore e del salto nel sistema C128 che include la descrizione del nome, indirizzo e funzione delle routine del sistema operativo Kernal.

VETTORI DEL SISTEMA C128

1.$FFFB SYSTEM	**;vettore del sistema operativo (RAM1)**
2.$FFFA NMI	**;vettore del processore NMI**
3.$FFFC RESET	**;vettore del processore RESET**
4.$FFFE IRQ	**;vettore del processore IRQ/BRK**

CHIAMATE DELLA TABELLA DI SALTO KERNAL STANDARD CBM

1.$FF81 JMP CINT	**;iniz l'editor dello schermo e le unità**
2.$FF84 JMP IOINIT	**;iniz le unità di I/O**
3.$FF87 JMP RAMTAS	**;iniz RAM e buffer**
4.$FF8A JMP RESTOR	**iniz i vettori indiretti (sistema)**
5.$FF8D JMP VECTOR	**;iniz i vettori indiretti (utente)**
6.$FF90 JMP SETMSG	**;messaggi kernal on/off**
7.$FF93 JMP SECND	**;seriale: manda SA dopo LISTN**
8.$FF96 JMP TKSA	**;seriale: manda SA dopo TALK**
9.$FF99 JMP MEMTOP	**;pone/legge la parte alta del sistema RAM**
10.$FF9C JMP MEMBOT	**;pone/legge la parte bassa del sistema RAM**
11.$FF9F JMP KEY	**;analizza la tastiera**
12.$FFA2 JMP SETTMO	**;(riservato)**
13.$FFA5 JMP ACPTR	**;seriale: input del byte**
14.$FFA8 JMP CIOUT	**;seriale: output del byte**
15.$FFAB JMP UNTLK	**;seriale: manda non parlare**
16.$FFAE JMP UNLSN	**;seriale: manda non ascoltare**
17.$FFB1 JMP LISTN	**;seriale: manda ascolta**
18.$FFB4 JMP TALK	**;seriale: manda parla**
19.$FFB7 JMP READSS	**;legge l'I/O del byte di stato**
20.$FFBA JMP SETLFS	**;pone i canali LA, FA, SA**
21.$FFBD JMP SETNAM	**;pone i puntatori del nome del file**
22.$FFC0 JMP(IOPEN)	**;apre file logico**
23.$FFC3 JMP(ICLOSE)	**;chiude file logico**
24.$FFC6 JMP(ICHKIN)	**;pone il canale di input**
25.$FFC9 JMP(ICKOUT)	**;pone il canale di output**
26.$FFCC JMP(ICLRCH)	**;ristabilisce i canali di default**
27.$FFCF JMP(IBASIN)	**;input dal canale**
28.$FFD2 JMP(IBSOUT)	**;output verso il canale**
29.$FFD5 JMP LOAD	**;carica dal file**
30.$FFD8 JMP SAVE	**;salva nel file**

31.$FFDB JMP SETTIM	;pone il clock interno
32.$FFDE JMP RDTIM	;legge il clock interno
33.$FFE1 JMP(ISTOP)	;analizza il tasto STOP
34.$FFE4 JMP(IGETIN)	;legge i dati bufferizzati
35.$FFE7 JMP(ICLALL)	;chiude tutti i file ed i canali
36.$FFEA JMP UDTIM	;incrementa il clock interno
37.$FFED JMP SCRORG	;ottiene la misura della finestra corrente
38.$FFF0 JMP PLOT	;legge/pone la posizione del cursore
39.$FFF3 JMP IOBASE	;legge l'indirizzo base del blocco di I/O

NUOVE CHIAMATE DELLA TABELLA DI SALTO KERNAL DEL C128

1.$FF47 JMP SPIN SPOUT	;posiziona le porte seriali veloci per l'I/O
2.$FF4A JMP CLOSE ALL	;chiude tutti i file di un'unità
3.$FF4D JMP C64MODE	;riconfigura il sistema come un C64
4.$FF50 JMP DMA CALL	;invia il comando all'unità DMA
5.$FF53 JMP BOOT CALL	;fa il boot del programma di caricamento da dischetto
6.$FF56 JMP PHOENIX	;iniz le cartucce delle funzioni
7.$FF59 JMP LKUPLA	;cerca le tabelle per LA determinato
8.$FF5C JMP LKUPSA	;cerca le tabelle per SA determinato
9.$FF5F JMP SWAPPER	;passa da 40 ad 80 colonne
10.$FF62 JMP DLCHR	;iniz la RAM ad 80 colonne
11.$FF65 JMP PFKEY	;programma un tasto funzione
12.$FF68 JMP SETBNK	;pone il banco per le operazioni di I/O
13.$FF6B JMP GETCFG	;riconosce il dato MMU per il banco determinato
14.$FF6E JMP JSRFAR	;gosub un altro banco
15.$FF71 JMP JMPFAR	;goto un altro banco
16.$FF74 JMP INDFET	;LDA (fetvec),Y da qualsiasi banco
17.$FF77 JMP INDSTA	;STA (stavec),Y in qualsiasi banco
18.$FF7A JMP INDCMP	;CMP (cmpvec),Y in qualsiasi banco
19.$FF7D JMP PRIMM	;stampa le utility immediate

Figura 14-1. Routine Kernal richiamabili dall'Utente

La Figura 14-2 lista le convenzioni usate nella descrizione di ogni subroutine Kernal. La Figura è seguita da descrizioni dei vettori del sistema del C128 e delle subroutine Kernal. In ogni descrizione sono inclusi il nome della subroutine, l'indirizzo di chiamata, le routine di preparazione di cui si ha bisogno (se ne avete bisogno), i registri interessati, i codici di errore associati ad ogni routine, come usarli ed un esempio di ogni chiamata di subroutine Kernal.

CONVENZIONI DELLE ROUTINE KERNAL RICHIAMABILI DA PARTE DELL'UTENTE

Indirizzo di Chiamata: Questo è l'indirizzo di chiamata della routine Kernal, dato in esadecimali.
Nome della Funzione: Nome della routine Kernal.
Registro: I registri, la memoria ed i flag listati sotto questo titolo vengono usati per passare i parametri verso e dalle routine Kernal.
Routine di preparazione: Alcune routine Kernal richiedono che i dati vi siano posti da routine di preparazione prima che la routine voluta possa essere richiamata. Le routine necessarie sono listate qui.
Riporti di Errore: Un ritorno da una routine Kernal con il riporto indica che è stato incontrato un errore nel procedimento. L'accumulatore conterrà il numero dell'errore.
Codici di Errore: Sotto c'è una lista di messaggi di errori che possono avvenire quando si usano le routine Kernal. Se capita un errore durante una routine Kernal, viene settato il bit di riporto dell'accumulatore ed il numero del messaggio dell'errore viene riportato nell'accumulatore.

> **NOTA:** Alcune routine Kernal di I/O non utilizzano questi codici per i messaggi di errore. Invece, gli errori vengono identificati usando la routine Kernal READST.

NUMERO	SIGNIFICATO
0	**Routine terminata con il tasto STOP**
1	**Troppi file aperti**
2	**File già aperto**
3	**File non aperto**
4	**File non trovato**
5	**Unità non presente**
6	**Il file non è un file di input**
7	**Il file non è un file di output**
8	**Manca il nome del file**
9	**Numero dell'unità non valido**
41	**Errore nella lettura del file**

Registri Coinvolti: Tutti i registri, la memoria ed i flag usati dalla routine Kernal sono qui listati.
Esempi: È listato un esempio per ogni routine Kernal.
Descrizione: Viene data una breve spiegazione della funzione della routine Kernal.

Figura 14-2. Convenzioni delle Routine Kernal richiamabili da parte dell'Utente.

VETTORI DEL SISTEMA C128

I vettori listati qui sotto, con l'eccezione del vettore SYSTEM, sono localizzati non solo nella ROM del sistema ma in ogni banco della RAM. L'inizio e la fine dei gestori di interruzione del sistema si trovano nella pagina in alto ($FFxx) di tutte le configurazioni della memoria. La ragione è semplice: un'interruzione può avvenire in qualsiasi minuto, da qualsiasi configurazione della memoria. I registri e la configurazione della memoria devono essere conservati prima di portare il sistema operativo al punto di elaborare l'interruzione. Essi devono venire ripristinati in modo simile prima che il controllo venga infine riportato al codice interrotto. Notate che i vettori del sistema sono salti indiretti e di solito non vengono richiamati dall'utente poichè terminano con un'istruzione RTI, non con un'istruzione RTS. In altre parole, essi elaborano eventi interrotti, non richiami delle subroutine.

1. $FFF8 SYSTEM ;vettore del sistema operativo (RAM 1)
   Il vettore **SYSTEM** e la stringa **KEY** di accompagnamento forniscono applicazioni software in modo da riprendere il controllo del sistema dopo un ripristino dell'hardware. Con questo vettore, il software può essere protetto da una situazione altrimenti irrecuperabile. La stringa KEY fornisce la distinzione tra un ripristino "caldo" ed un inserimento "freddo". Se il sistema è appena stato inserito, la stringa KEY manca e così il vettore SYSTEM non è valido; viene installato il **CBM** KEY ed il vettore SYSTEM è posto in modo C128. Se il sistema era stato ripristinato (cioè, era stato trovato in KEY), il vettore del sistema viene mosso in una RAM comune in $02 e viene compiuto un JMP indiretto. Nella maggior parte dei casi, l'utente deve chiamare solo IOINIT prima di riprendere il controllo. Lo schema nella RAM-1 è:

   $FFF5 "C" ($43)
   $FFF6 "B" ($42)
   $FFF7 "M" ($4D)
   $FFF8 SYSTEM vettore basso
   $FFF9 SYSTEM vettore alto

   Per esempio, supponete che un programmatore cominci ad eseguire un' "operazione" molto complessa e ripetitiva usando il Monitor incorporato. Realizzandosi la possibilità di perdere il controllo a causa di un guasto, il programmatore potrebbe dirigere nuovamente il vettore SYSTEM al Monitor stesso e così riprendere il controllo con una perdita minima della RAM, semplicemente premendo il tasto RESET. Per fare questo, il programmatore inserirà:

```
a. >1FFF8    00 13:dirigere SYSTEM
b. A 1300    JSR $FF84           :richiamare IOINIT
             JMP $B000           :richiamare Monitor
```

2. $FFFA NMI ;processore vettore NMI
   Il vettore di **Interruzione Non Mascherabile (NMI)** è attivato tutte le volte che il pin (piedino) del processore NMI trova un fronte discendente di un impulso. Ci sono due possibili origini di NMI in condizioni normali: il tasto **RESTORE** e RS-232 I/O. Se avviene un NMI, il sistema operativo disabilita IRQ, salva i registri e la configurazione della memoria corrente nello stack, porta la configurazione del sistema (ROM, I/O, RAM 0) nel contesto, ed esegue un salto indiretto fino al vettore della RAM localizzato in $318. Il gestore (handler) del sistema NMI azzera il registro ICR del CIA-2, dal quale determina l'origine dell'interruzione. Se viene dal timer A, il controllo passa al transceiver (terminale che può ricevere e trasmettere messaggi) dell'RS-232. Se non è così, viene assunto il tasto **RESTORE** e per sicurezza, viene scelta la convenzione CBM che richiede che il tasto **STOP** venga simultaneamente premuto. Se viene premuto il tasto STOP, tutti i vettori indiretti del sistema vengono ripristinati, vengono richiamati IOINIT e CINT, e viene preso il SYSTEM-VECTOR (da non confondere con il vettore SYSTEM!). Il controllo viene riportato al ripristino dei registri e della configurazione della memoria. NMI può essere disabilitato causando un NMI iniziale dal timer A, ma mai con la lettura di ICR perchè lo cancelli, collegando così il segnale NMI a massa.
   Il software di applicazione può intercettare un evento NMI modificando uno dei due vettori della RAM menzionati sopra. Il vettore indiretto NMI in $318 nella RAM comune passerà il controllo tutte le volte che capita un NMI. Il SYSTEM-VECTOR, localizzato in $A00 nella RAM 0, passerà il controllo dopo aver riconosciuto e gestito STOP/RESTORE.
   Per esempio, supponete che avvenga una situazione simile a quella illustrata prima per il vettore SYSTEM. Per riportare il controllo al Monitor tutte le volte che viene riconosciuto lo STOP/RESTORE inserite:

   a. >A00 00 B0: inviare SYSTEM-VECTOR al Monitor

   In modo simile, per eseguire un'operazione tutte le volte che avviene un NMI (cioè solo RESTORE), usate NMI indiretto. Per esempio, incrementate il registro del colore di contorno del VIC tutte le volte che premete **RESTORE** (o causate qualsiasi altro NMI). Inserite:

   a. >318 00 13 :inviare l'indiretto a $1300
   b. A 1300 INC $D020 :cambiare il colore di contorno
   JMP $FF33: ritornare dall'interruzione

3. $FFFC RESET ;processore vettore RESET
   Il vettore **RESET** del processore è attivato tutte le volte che il segnale RESET del sistema è basso. Esso è basso al momento dell'attivazione, e diventa basso premendo il tasto RESET. Questo segnale ha effetto non solo sul processore ma anche sulla maggior parte delle unità di I/O che si trovano nel sistema. Infatti, RESET è il segnale di controllo del processore che è in comune ai due processori (8502 e Z80) del C128. Lo Z80 ha il controllo iniziale (per default) del sistema mentre l'8502 viene tenuto in stato di attesa. Quando l'8502 parte infine dopo un reset, la routine di inizializzazione START Kernal riceve sempre il controllo ed esegue immediatamente le seguenti azioni:

1. Porta la mappa del sistema nel contesto.
2. Disabilita gli IRQ.
3. Ripristina il puntatore dello stack del processore.
4. Azzera il modo decimale.
5. Inizializza l'MMU.
6. INSTALLA il codice della RAM Kernal.

Fino a questo punto non ci sono disposizioni per un codice dell'utente. Le due routine successive nel percorso di inizializzazione cercano realmente un codice dell'utente installato:

7. SECURE: Controlla ed inizializza il vettore SYSTEM.
8. POLL: Controlla una cartuccia ROM.

POLL per prima cosa analizza la presenza di qualsiasi cartuccia installata del C64. Vengono riconosciute quando sia il segnale **GAME** che **EXROM** divengono bassi. Se questo accade, viene eseguito GO64 (consultare le entrate a salto del Kernal per i dettagli). Il poll delle cartucce C64 in realtà è superfluo a questo punto poichè il processore Z80, che è inserito inizialmente, ha già fatto questo. Allora POLL analizza la presenza di cartucce del C128 e delle funzioni della ROM. Esse vengono riconosciute dalla esistenza del tasto C= in uno qualsiasi dei quattro slot delle funzioni della ROM (due interni, due esterni) e subiscono il poll (vengono interrogati) nell'ordine esterno basso (16 o 32KB), esterno alto (16KB), interno basso (16 o 32KB), interno alto (16KB). L'intero formato è:

$x000 → partenza a freddo
$x003 → partenza a caldo (non usata)
$x006 → ID.($01-$FF)
$x007 → "CBM" stringa chiave
dove x = $8--- o $C---.

L'ID di ogni cartuccia del C128 trovata è inserito nella Tabella dell'Indirizzo Fisico (PAT) localizzata in SAC1-SAC4. ID non deve essere zero. Le cartucce possono riconoscersi tra loro esaminando la PAT per ID particolari. Un ID uguale a 1 indica una cartuccia auto iniziante e la sua partenza a freddo sarà richiamata immediatamente. Tutte le altre saranno chiamate più tardi (consultare il salto PHOENIX), così come quelle auto inizianti che RTS vuole interrogare (poll). Una cartuccia può determinare dove è installata esaminando CURBNK, localizzato in $ACO. Poichè è possibile per una cartuccia essere chiamata quando le interruzioni sono inserite, si raccomanda la seguente deviazione dal formato di cui sopra (la partenza a caldo non viene mai richiamata dal sistema):

```
$x000 SEI
$x001 JMP STARTUP
$x004 NOP
$ x005 NOP
```

Il saldo dell'inizializzazione del C128 è:

9. IOINIT: Inizializza le unità di I/O.
10. Controlla i tasti STOP AND C= .
11. RAMTAS: Inizializza il sistema RAM.
12. RESTOR: Inizializza gli indiretti del sistema.
13. CINT: Inizializza i video.
14. Abilita gli IRQ (tranne i sistemi stranieri).
15. Smistamento.

IOINIT è forse la maggiore funzione dell'handler del Reset. Essa inizializza sia i CIA (i timer, la tastiera, la porta seriale, l'ingresso dell'utente, la cassetta) che la porta dell'8502 (tastiera, cassetta, banco VIC). Essa distingue un sistema PAL da uno NTSC e pone il PALCNT ($A03) se è PAL. Le unità VIC, SID e 8563 vengono inizializzate, incluso il caricamento delle definizioni dei caratteri nella RAM video dell'8563 (se necessario). La sorgente del sistema IRQ 60Hz (il raster del VIC) viene attivata. IOINIT è richiamato dall'utente con la tabella di salto.
Durante l'inizializzazione, l'utente può digitare alcuni tasti come mezzo per selezionare un modo operativo. Un tasto che viene scelto è il tasto Commodore C= , che indica che si desidera il modo C64. Mentre questo tasto veniva esaminato molto prima dallo Z80 per rendere più veloce il deviatore in modo C64, viene effettuato un controllo ridondante. L'unico altro tasto analizzato in questo momento è il tasto **STOP,** che segnala una richiesta da parte dell'utente di inserirsi nelle utility del Monitor. Notate che il controllo non passa dal processo di inizializzazione in questo momento; il Kernal ha bisogno di sapere se si può saltare **RAMTAS.** Solo se il tasto **STOP** è premuto e questo era un ripristino "caldo" (al contrario di una partenza a "freddo") allora RAMTAS può essere saltato.
RAMTAS cancella tutte le pagine zero della RAM, assegna i buffer della cassetta e dell'RS-232, pone i puntatori in cima ed in fondo alla RAM del sistema (RAM 0), ed installa il SYSTEM-VECTOR (trattato prima sotto NMI) che punta la partenza a freddo del BASIC. Infine pone un flag, DEJAVU ($A02), per indicare alle altre routine che la RAM del sistema è stata inizializzata. Questa è la differenza tra un sistema "freddo" ed uno "caldo". Se DEJAVU contiene $A5, il sistema è "caldo" ed il SYSTEM-VECTOR è valido. Molti codici di debugging del programmatore necessitano una riattivazione dopo un arresto o un guasto del sistema via **RESET** ma non vogliono che la RAM sia cancellata. Questo è il motivo per cui viene cercato il tasto **STOP**, RAMTAS viene saltato e viene selezionato il Monitor (piuttosto che il BASIC). RAMTAS è richiamabile da parte dell'utente con la tabella di salto.
RESTOR inizializza i vettori indiretti del Kernal. Questo deve essere fatto prima che molte routine del sistema funzionino. Le applicazioni che completano il sistema operativo via "inserimenti" (wedges) devono essere installati dopo che siano stati inizializzati. RESTOR è richiamabile da parte dell'utente dalla tabella di salto (consultare anche il richiamo del VETTORE).
CINT è la routine di inizializzazione dell'Editor. Vengono preparati sia il modo del video a 40 colonne che quello del video ad 80 colonne, i vettori indiretti dell'editor vengono installati, vengono assegnate le definizioni dei tasti programmabili ed i tasti 40/80 vengono analizzati per determinare la modalità del video. CINT è anche un'entrata della tabella di salto.

Infine, gli IRQ sono abilitati ed il controllo passa sia all'inizializzazione del BASIC, che al codice GO64, che al Monitor ML. Il BASIC chiamerà la routine PHOENIX del Kernal alla conclusione della sua inizializzazione, che chiamerà qualsiasi cartuccia installata del C128 (qualsiasi ID) e tenterà di fare l'autoboot di un'applicazione dal dischetto.
Un byte di stato dell'inizializzazione, INIT-STATUS ($A04), segna il progresso del processo di inizializzazione. Viene cancellato automaticamente all'inizio del codice Reset e, quando sono completati degli specifici stadi, viene posto un bit particolare. Il tracciato è il seguente:

B7 → caratteri dell'8563 installati
B6 → eseguito CINT
B0 → BASIC inizializzato

Qualsiasi richiamo di IOINIT, Reset compreso, non risulterà nell'inizializzazione della RAM dell'8563 se viene posto B7. In modo simile, CINT non inizializzerà le tabelle di miglioramento delle matrici della tastiera e le definizioni programmabili dei tasti se viene posto B6. Ecco come NMI, per esempio, può chiamare IOINIT e CINT senza distruggere le predisposizioni dell'utente. Infine, l'inizializzazione del BASIC deve essere completa prima che venga posto B0. Ciò determina se l'handler IRQ, per esempio, possa chiamare le routine IRQ in BASIC. Notate che la sequenza seguente di eventi dovrebbe essere portata a termine da un programmatore in BASIC per ristabilirsi da un guasto via **RESET:**

1. Tenere premuto il tasto **STOP** per entrare nel Monitor.
2. Premere e lasciare il tasto **RESET.**
3. Lasciare il tasto **STOP.**
4. Inserire: >A04 C1: riabilitare BASIC IRQ.
5. Inserire: X : uscire dal Monitor verso il BASIC.

Questa sequenza è necessaria perchè INIT-STATUS era stato cancellato dal reset, e l'inizializzazione in BASIC era stata saltata, resettando B0.
Se non fosse stato posto B0, le routine IRQ BASIC come gli handler SOUND, PLAY e SPRITE non avrebbero funzionato, risultando di solito in un "arresto" apparente.

4. $FFFE IRQ ;vettore IRQ/BRK del processore
Questo vettore hardware è preso tutte le volte che il pin **IRQ** del processore viene posto basso, o che il processore esegue un'istruzione **BRK.** Per un'operazione corretta un IRQ deve avvenire sessanta volte al secondo [NTSC (60Hz) non presenta alcun problema, ma si devono operare delle correzioni per i sistemi PAL (50Hz)].
La fonte usuale di IRQ nel C128 è il raster del VIC, che è liberato durante l'inizializzazione del sistema da IOINIT. Se capita un IRQ o un BRK, il sistema operativo salva i registri e la configurazione corrente della memoria nello stack e porta il banco del sistema (ROM, I/O, RAM 0) nel contesto. Al momento dell'interruzione lo stato del processore viene letto dallo stack del sistema per determinare se l'interruzione era un IRQ o un BRK. Il C128 usa i seguenti codici per fare ciò:

```
TSX             ; ottiene il puntatore dello stack
LDA $105,X      ; recupero lo stato del processore
AND #$10        ; esamina il flag del BRK
```

Se viene posto il flag del BRK, il controllo passa al Monitor ML attraverso il vettore indiretto BRK in $316, che di solito punta all'entrata BREAK del Monitor. Qui vengono recuperati e visualizzati il contatore del programma (PC), lo stato del processore, i registri, la configurazione della memoria ed il puntatore dello stack.
Se il flag del BRK è 0, viene assunto un IRQ ed il controllo passa attraverso il vettore indiretto IRQ in $314, di solito all'handler IRQ del sistema. Allora vengono eseguiti i seguenti procedimenti nell'ordine mostrato:

1. IRQ è disabilitato.
2. Editor: gestore (handler) di divisione dello schermo.
3. Editor: azzera il raster del VIC IRQ.
4. IRQ è abilitato.
5. Editor: analisi della tastiera.
6. Editor: lampeggio del cursore VIC.
7. Kernal: "jiffie" clock.
8. Kernal: interruttori della cassetta.
9. Kernal: cancella CIA-1 ICR.
10. BASIC : sprite, suoni, ecc.
11. Ritorno dall'interruzione.

Come indicato nella descrizione precedente, il sistema operativo del C128 usa molto l'IRQ. In particolare, l'handler della divisione dello schermo dell'Editor ha delle esigenze piuttosto rigide che i programmatori devono riconoscere e sistemare. Uno schermo tutto-testo o uno schermo pieno di grafici non presentano particolari problemi, ma un testo diviso ed uno schermo con alcuni grafici richiedono il doppio della frequenza di IRQ (due ogni sessantesimo di secondo). Così, l'Editor (e di conseguenza, le applicazioni dipendenti da IRQ) devono distinguere tra l'IRQ 60 Hz "principale" e l'IRQ "di mezzo". Solo durante l'IRQ principale verranno eseguite tutte le azioni listate sopra; l'IRQ di mezzo viene usato solo dall'Editor per dividere lo schermo. La routine IRQ dell'Editor pone il flag del riporto per designare un IRQ principale. Inoltre, non c'è limite alle richieste di tempo di uno schermo diviso. I programmatori dovrebbero notare il modo in cui l'Editor usa il VIC durante gli IRQ ed evitare I/O diretti del VIC se le operazioni dello schermo dell'Editor sono abilitate. C'è un byte di flag chiamato **GRAPHM,** localizzato in $D8, che può essere posto in $FF per disabilitare l'uso del VIC da parte dell'Editor.
L'Editor è anche responsabile dell'analisi della tastiera del C128. I programmatori dovrebbero notare che **SCNKEY** ha due salti indiretti, **KEYLOG** e **KEYPUT,** che usa durante le esecuzioni. La tastiera è controllata via CIA-1 PRA, PRB, il registro VIC #47 (matrice estesa del tasto) e la porta 8502 (bit 6 = CAPS LOCK). La routine SCNKEY è richiamata dalla tabella di salto.
L'andamento delle routine IRQ fino al richiamo del BASIC si spiega da solo. Il clock del software Kernal è mantenuto da UDTIM, che è nella tabella di salto. Il procedimento IRQ fa un ultimo richiamo al BASIC-IRQ ($4006), ma solo se

INIT-STATUS del bit 0 viene posto per indicare il BASIC esso è pronto a gestire gli IRQ (di questo si è trattato prima nella sezione RESET). Il BASIC-IRQ sfrutta molto il VIC ed il SID, e si dovrebbero prendere le stesse precauzioni rispetto all'I/O diretto del VIC e del SID come per l'Editor. Il BASIC-IRQ utilizza anche un byte di trattenimento chiamato IRQ-WRAP-FLAG. Esso viene normalmente usato dal sistema per bloccare i richiami IRQ che hanno subito un IRQ, ma possono essere posti dall'utente con l'effetto di disabilitare l'handler del BASIC-IRQ. Come alternativa potete azzerare il bit 0 del byte INIT-STATUS come menzionato prima ed ottenere lo stesso risultato.
L'uso dell'indiretto IRQ ($314) per mezzo di un'applicazione di solito richiede più attenzione rispetto alla maggior parte degli altri inserimenti (wedge) per parecchie ragioni. La probabilità che avvenga un IRQ mentre viene installato un inserimento è molto grande, esiste la possibilità che l'utente o qualche altro software abbia già inserito il vettore, e di solito è meglio per le funzioni del sistema IRQ che continuino normalmente (esempio, l'analisi dei tasti) piuttosto che sostituirli completamente con i vostri (come abbiamo fatto con gli esempi NMI). Gli esempi seguenti eseguono queste funzioni e anche il mascheramento di tutto tranne che dell'IRQ principale. Questo esempio cambia i tasti **40/80** in tasti SLOW/FAST:

```
A 1302 BIT  $D011      ;test VIC reg 17
       BPL  $1324      ;salta se l'IRQ è errato
       LDA  $D505      ;preparazione
       ORA  #$80       ;b7 di MMU MCR
       STA  $D505      ;regolazione per l'input
       LDA  $D011      ;assume una predisposizione FAST (veloce)
       AND  #$6F       ;azzera il VIC (RC8=0)
       LDX  #$01       ;2MHz
       BIT  $D505      ;test dei tasti 40/80
       BPL  $131E      ;salta se giù (FAST)
       ORA  #$10       ;ripone il VIC
       DEX             ;1MHz
  131E STX  $D030      ;pone la velocità
       STA  $D011      ;pone il blank del bit
  1324 JMP  ($1300)    ;continua il sistema IRQ
```

Ora abbiamo bisogno di una piccola routine per inserire realmente il nostro codice nel sistema IRQ. Il codice seguente salva il vettore corrente IRQ perchè il nostro gestore (handler) di cui sopra possa uscire e sostituisce un puntatore del nostro codice:

```
A 1400 SEI             ;previene le interruzioni
       LDA  $314       ;ottiene il corrente lsb IRQ
       STA  $1300      ;e lo salva
       LDA  $315       ;ottiene il corrente msb IRQ
       STA  $1301      ;e lo salva
       LDA  #$02       ;ottiene il nostro lsb IRQ
       STA  $314       ;e lo sostituisce
       LDA  #$13       ;ottiene il nostro msb IRQ
       STA  $315       ;e lo sostituisce
       CLI             ;riabilita il procedimento IRQ
       RTS
```

Abilitate l'inserimento digitando **J 1400**, ecco tutto. Premendo il tasto Lock **40/80** si entra in modo FAST; lasciandolo si torna in SLOW, e l'analisi dei tasti, etc., continua a funzionare. Notate, tuttavia, in questo schermo diviso il nostro codice toglie la sincronizzazione, favorendo una visualizzazione non bella. Ci sono in realtà solo tre cose a cui stare attenti quando si opera con il sistema IRQ del C128: primo, assicuratevi di mantenere il valore di comparazione del raster sullo schermo per mantenere l'evento IRQ (il modo migliore è di mantenere l'RC8 a zero come nell'esempio sopra); secondo, non tentate mai di accedere all'8563 durante un IRQ se c'è una possibilità che esso sia in uso; infine, siate sicuri che l'origine dell'IRQ sia stata cancellata.

CHIAMATE STANDARD DEL KERNAL CBM

Le chiamate di sistema seguenti formano l'insieme delle chiamate di sistema standard CBM per il Personal Computer Commodore 128. Parecchie chiamate, tuttavia, funzionano in modo un pò diverso o possono richiedere delle disposizioni leggermente diverse rispetto al modo C64. Questo è stato necessario per sistemare delle funzioni specifiche del sistema, l'Editor della finestra a 40/80 colonne ed i mezzi per la memoria in cui si depositano i dati. Come per tutti i richiami Kernal, la configurazione del sistema (ROM alta, RAM-0 e I/O) deve essere attiva al momento della chiamata.

1. $FF81 CINT ;inizializza l'editor dello schermo e le unità

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
	.X usato
	.Y usato
Memoria:	iniz Editor RAM
	iniz Editor I/O
Flag:	nessuno

ESEMPIO:

```
SEI
JSR $FF81          ;inizializza l'editor dello schermo
CLI
```

CINT è la routine di inizializzazione dell'Editor. Vengono preparati entrambi i modi di visualizzazione a 40 ed 80 colonne, vengono installati i vettori indiretti dell'editor, vengono assegnate le definizioni programmabili dei tasti ed analizzati i tasti **40/80** per determinare la modalità del video. CINT pone il banco VIC ed il banco nybble VIC, abilita il carattere ROM, ristabilisce il volume SID, pone sia lo schermo a 40 colonne che quello ad 80 e li cancella. L'unica cosa che non fa, che è compito dell'Editor, è l'inizializzazione dell'I/O, di cui si ha bisogno per l'IRQ (analisi dei tasti, lampeggio del cursore VIC, modi di divisione dello schermo), linee chiave, colori di fondo dello schermo, etc. (consultare IOINIT). Poichè CINT aggiorna i vettori indiretti dell'Editor che vengono usati durante il procedimento IRQ, voi dovreste disabilitare gli IRQ prima di richiamarli. CINT utilizza il byte di stato INIT-STATUS ($A04) come segue:

```
$A04 bit 6 = 0 →  Inizializzazione completa.
                  (pone INIT-STATUS bit 6)
           = 1 →  Inizializzazione parziale.
                  (non sono inizializzati i puntatori delle matrici dei tasti)
                  (non sono inizializzate le definizioni dei tasti del programma)
```

CINT è anche un'entrata Editor della tabella di salto ($C000).

2. $FF84 IOINIT ;iniz delle unità I/O

```
PREPARAZIONE:
    Registri:        nessuno
    Memoria:         mappa del sistema
    Flag:            nessuno
    Chiamate:        nessuna

RISULTATI:
    Registri:        .A usato
                     .X usato
                     .Y usato
    Memoria:         inizializza l'I/O
    Flag:            nessuno
```

ESEMPIO:

```
    SEI
    JSR $FF84        ;inizializza l'I/O del sistema
    CLI
```

IOINIT è forse la funzione maggiore dell'handler del Reset. Essa inizializza sia la porta CIA (timer, tastiera, porta seriale, ingresso dell'utente, cassetta) che la porta dell'8502 (tastiera, cassetta, banco VIC). Distingue un sistema PAL da uno NTSC e se è PAL pone PALCNT ($A03). Le unità VIC, SID e 8563 vengono inizializzate, incluso il caricamento delle definizioni dei caratteri nella RAM video dell'8563 (se necessario). La generazione delle IRQ di sistema a 60 Hz, il raster del VIC, inizia (in attesa che gli IRQ siano cancellati). IOINIT utilizza il byte di stato INIT-STATUS ($A04) come segue:

$A04 bit 7 = 0 → Inizializzazione completa.
(pone INIT-STATUS bit 7)
= 1 → Inizializzazione parziale.
(escluse le definizioni dei caratteri dell'8563)

Dovete essere sicuri che gli IRQ siano disabilitati prima di chiamare IOINIT per evitare interruzioni mentre le varie unità di I/O vengono inizializzate.

3. $FF87 RAMTAS ;iniz la RAM ed i buffer

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
	.X usato
	.Y usato
Memoria:	inizializza la RAM
Flag:	nessuno

ESEMPIO:

```
JSR $FF87        ;inizializza la RAM del sistema
```

RAMTAS azzera tutte le pagine zero della RAM, assegna la cassetta ed i buffer RS-232, pone i puntatori in cima ed in fondo alla RAM del sistema (RAM 0) e punta il SYSTEM-VECTOR ($A00) alla partenza a freddo del BASIC ($4000). Infine, pone un flag, DEJAVU ($A02), per indicare alle altre routine che la RAM del sistema è stata inizializzata e che il SYSTEM-VECTOR è valido. Si dovrebbe notare che la routine RAMTAS del C128 non prova assolutamente la RAM.

4. $FF8A RESTOR ;iniz gli indiretti del Kernal

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
	.X usato
	.Y usato
Memoria:	ripristinati gli indiretti del Kernal
Flag:	nessuno

ESEMPIO:

```
SEI JSR $FF8A      ;ripristina gli indiretti del Kernal
CLI
```

RESTOR ripristina i valori per default di tutti i vettori indiretti del Kernal dalla lista della ROM Kernal. Non tocca nessun altro vettore, come quelli usati dall'Editor (consultate CINT) e dal BASIC. Poichè è possibile che un'interruzione (IRQ o NMI) capiti durante l'aggiornamento dei vettori indiretti interrotti, dovreste disabilitare le interruzioni prima di chiamare il RESTOR. Consultate anche la chiamata VECTOR.

5. $FF8D VECTOR ;iniz o copia gli indiretti

PREPARAZIONE:

Registri:	.X = indirizzo (basso) della lista dell'utente
	.Y = indirizzo (alto) della lista dell'utente
Memoria:	mappa del sistema
Flag:	.C = 0 → carica i vettori Kernal
	.C = 1 → copia i vettori Kernal
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
	.Y usato
Memoria:	come per la chiamata
Flag:	nessuno

ESEMPIO:

```
LDX #salva-basso
LDY #salva-alto
SEC
JSR $FF87          ;copia gli indiretti in "salva"
```

VECTOR legge o scrive i vettori indiretti della RAM Kernal. Chiamando VECTOR con il riporto di stato memorizza i contenuti correnti dei vettori indiretti nell'indirizzo della RAM passato nei registri .X e .Y (al banco della

RAM corrente). Chiamando VECTOR con il riporto di stato cancellato aggiorna i vettori indiretti del Kernal dalla lista dell'utente passati nei registri .X e .Y (dal banco della RAM corrente). Le interruzioni (IRQ e NMI) dovrebbero essere disabilitate quando si aggiornano gli indiretti. Consultate anche la chiamata di RESTOR.

6. $FF90 SETMSG ;messaggi Kernal accesi/spenti

PREPARAZIONE:

Registri:	.A = controllo del messaggio
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	nessuno
Memoria:	MSGFLG ($D9) aggiornata
Flag:	nessuno

ESEMPIO:

```
LDA #0
JSR $FF90          ;spegne tutti i messaggi Kernal
```

SETMSG aggiorna il byte del flag del messaggio Kernal MSGFLG ($9D) che determina se gli errori del sistema e/o i messaggi di controllo verranno visualizzati. Il BASIC di solito disabilita sempre i messaggi di errore e disabilita i messaggi di controllo in modo Run. Notate che i messaggi di errore Kernal non sono quelli prolissi stampati dal BASIC, ma per esempio, semplicemente il messaggio **I/O ERROR** # che vedete quando siete sul Monitor. Esempi di messaggi di controllo Kernal sono **LOADING**, **FOUND** e **PRESS PLAY ON TAPE**. I bit di controllo del MSGFLG sono:

MSGFLG bit 7 = 1 → i messaggi di CONTROLLO
bit 6 = 1 → abilita i messaggi di ERRORE

7. $FF93 SECND ;seriale: manda SA dopo LISTN

PREPARAZIONE:

Registri:	.A = SA (indirizzo secondario)
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	LISTN

RISULTATI:

Registri:	.A usato
Memoria:	STATUS ($90)
Flag:	nessuno

ESEMPIO:

```
LDA #8
JSR $FF81        ;LISTN unità 8
LDA #15
JSR $FF93        ;passa SA #15
```

SECND è una routine seriale a basso livello usata per inviare un indirizzo secondario (SA) ad un'unità che ha causato una chiamata della routine LISTN (consultate la chiamata Kernal LISTN). Un SA di solito è usato per fornire delle informazioni per predisposizioni ad un'unità prima che inizi la reale operazione di I/O dei dati. L'attenzione cessa dopo una chiamata di SECND. SECND non viene usato per inviare un SA ad un'unità che parla (TALKing) (consultate TKSA). (La maggior parte delle applicazioni dovrebbero utilizzare le routine I/O ad alto livello: consultate OPEN e CKOUT).

8. $FF96 TKSA ;seriale: manda SA dopo TALK

PREPARAZIONE:

Registri:	.A = SA (indirizzo secondario)
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	TALK

RISULTATI:

Registri:	.A usato
Memoria:	STATUS ($90)
Flag:	nessuno

ESEMPIO:

```
LDA #8
JSR $FFB4        ;TALK unità 8
LDA #15
JSR $FF93        ;passa SA #15
```

TKSA è una routine seriale a basso livello usata per inviare un indirizzo secondario (SA) ad un'unità a cui si comanda di parlare (TALK) (consultate la chiamata TALK del Kernal). Un SA di solito è usato per fornire informazioni di predisposizione ad un'unità prima di cominciare la reale operazione di I/O dati. (La maggior parte delle applicazioni dovrebbero usare le routine di I/O ad alto livello; consultate OPEN e CHKIN).

9. $FF99 MEMTOP ;pone/legge la parte alta della RAM del sistema

PREPARAZIONE:

Registri:	.X = lsb di MEMSIZ
	.Y = msb di MEMSIZ
Memoria:	mappa del sistema
Flag:	.C = 0 → pone la parte alta della memoria
	.C = 1 → legge la parte alta della memoria
Chiamate:	nessuna

RISULTATI:

Registri:	.X = lsb di MEMSIZ
	.Y = msb di MEMSIZ
Memoria:	MEMSIZ ($A07)
Flag:	nessuno

ESEMPIO:

```
SEC
JSR $FF99       ;ottiene la parte alta della RAM 0 dell'utente
DEY
CLC
JSR $FF99       ;lo abbassa di un blocco
```

MEMTOP è usato per leggere o per porre la parte alta della RAM del sistema, indicata da MEMSIZ ($A07). Questa chiamata è inclusa nel C128 per completezza, ma nè il Kernal nè il BASIC utilizzano il MEMTOP poichè ha poco significato nell'ambito della memoria a banchi del C128 (anche i buffer dell'RS-232 sono assegnati in modo permanente). Nondimeno, pone il riporto di stato per caricare MEMSIZ in .X e .Y e lo azzera per aggiornare il puntatore da .X e .Y. Notate che MEMSIZ fa riferimento solo alla RAM del sistema (RAM 0). Il Kernal pone inizialmente MEMSIZ in $FF00 (MMU ed il codice della RAM del Kernal iniziano qui).

10. $FF9C MEMBOT ;pone/legge la parte bassa della RAM del sistema

PREPARAZIONE:

Registri:	.X = lsb di MEMSTR
	.Y = msb di MEMSTR
Memoria:	mappa del sistema
Flag:	.C = 0 → pone la parte bassa della memoria
	.C = 1 → legge la parte bassa della memoria
Chiamate:	nessuna

RISULTATI:

Registri:	.X = lsb di MEMSTR
	.Y = msb di MEMSTR
Memoria:	MEMSTR ($A05)
Flag:	nessuno

ESEMPIO:

```
SEC
JSR $FF9C          ;ottiene la parte bassa della RAM 0 dell'utente
INY
CLC
JSR $FF9C          ;l'alza di un blocco
```

MEMBOT viene usato per leggere o porre la parte bassa della RAM del sistema, indicato da MEMSTR ($A05). Questa chiamata è inclusa nel C128 per completezza, ma nè il Kernal nè il BASIC utilizzano il MEMBOT poichè non ha molto significato nell'ambito della memoria a banchi del C128. Nondimeno, pone il riporto di stato per caricare MEMSTR in .X e .Y e lo azzera per aggiornare il puntatore da .X e .Y. Notate che MEMSTR fa riferimento solo alla RAM del sistema (RAM 0). Il Kernal pone inizialmente MEMSTR in $1000 (il testo in BASIC comincia qui).

11. $FF9F KEY ;analisi della tastiera

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	nessuno
Memoria:	buffer della tastiera
	flag della tastiera
Flag:	nessuno

ESEMPIO:

```
JSR $FF9F          ;analizza la tastiera
```

KEY è una routine dell'Editor che analizza l'intera tastiera del C128 (tranne il tasto **40/80**). Essa distingue tra i tasti ASCII, i tasti di controllo ed i tasti programmabili, ponendo i byte di stato della tastiera e gestendo il buffer della tastiera. Dopo aver decodificato il tasto, KEY gestirà tali funzioni come tasti bistabili, pause o ritardi e tasto di ripetizione. Esso viene di solito chiamato dal sistema operativo durante il procedimento IRQ 60Hz. Alla conclusione, KEY

lascia che l'hardware della tastiera guidi la linea del tasto sulla quale è localizzato il tasto **STOP.**
Si incontrano due salti indiretti della RAM durante l'analisi della tastiera: **KEYVEC** ($33A) e **KEYCHK** ($33C). KEYVEC (alias **KEYLOG)** viene preso tutte le volte che viene scoperto un tasto premuto, prima che il tasto in .A sia stato decodificato. KEYCHK viene preso dopo che il tasto è stato decodificato, proprio prima che venga posto nel buffer del tasto. KEYCHK porta il carattere ASCII in .A, il codice del tasto in .Y ed il tasto shift di stato in .X. Anche le matrici di decodifica della tastiera vengono indirizzate via vettori indiretti della RAM, localizzati in DECODE ($33E). La tabella seguente li descrive:

$33E Modo 1 → tasti normali
$340 Modo 2 → tasti **SHIFT**
$342 Modo 3 → tasti **COMMODORE**
$344 Modo 4 → tasti **CONTROL**
$346 Modo 5 → tasti **CAPS LOCK**
$348 Modo 6 → tasti **ALT**

La lista seguente descrive brevemente alcune delle variabili più importanti utilizzate o mantenute da KEY:

ROWS	$DC01	→	tasti di uscita della porta I/O
COLM	$DC00	→	tasti di pilotaggio C64 della porta I/O
VIC#47	$D02F	→	tasti nuovi di pilotaggio della porta I/O
8502 P6	$0001	→	tasto CAPS di lettura della porta I/O
NDX	$D0	→	indice del buffer della tastiera
KEYD	$34A	→	buffer della tastiera
XMAX	$A20	→	larghezza del buffer della tastiera
SHFLAG	$D3	→	tasto shift di stato
RPTFLG	$A22	→	abilita il tasto di ripetizione
LOCKS	$F7	→	disabilita il modo, la pausa

KEY si trova anche nella tabella di salto dell'Editor come SCNKEY in $C012.

12. $FFA2 SETTMO ;(riservato)

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	nessuno
Memoria:	TIMOUT ($A0E)
Flag:	nessuno

ESEMPIO:

```
LDA #valore
JSR $FFA2           ;aggiorna il byte TIMOUT
```

SETTMO non è usato nel C128 ma viene incluso per compatibilità e completezza. Viene usato nel C64 dalla cartuccia di comunicazione IEEE per disabilitare i timeout (sospensioni di tempo) dell'I/O.

13. $FFA5 ACPTR ;seriale: input di byte

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	TALK TKSA (se necessario)

RISULTATI:

Registri:	.A = byte del dato
Memoria:	STATUS ($90)
Flag	:nessuno

ESEMPIO:

```
JSR $FFA5           ;input di un byte dal bus seriale

STA dato
```

ACPTR è un'utilità seriale di I/O a basso livello per accettare un singolo byte dal TALKer del bus seriale corrente usando un handshaking (procedura di sincronizzazione delle comunicazioni prima del trasferimento dati) completo.
Per prepararsi a questa routine, un'unità deve prima essere stata stabilita come TALKer (consultate TALK) ed essere passata ad un indirizzo secondario se necessario (consultate TKSA). Il byte viene riportato in .A.
(La maggior parte delle applicazioni dovrebbero usare le routine di I/O a più alto livello; consultate BASIN e GETIN).

14. $FFA8 CIOUT ;seriale: output di byte

PREPARAZIONE:

Registri:	.A = byte del dato
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	LISTN SECND (se necessario)

RISULTATI:

Registri:	.A usato
Memoria:	STATUS ($90)
Flag:	nessuno

ESEMPIO:

```
LDA dato
JSR $FFA8        ;invia un byte via bus seriale
```

CIOUT è un'utilità seriale di I/O a basso livello per trasmettere un singolo byte al LISTNer del corrente bus seriale usando un handshaking completo. Per prepararsi a questa routine, un'unità deve prima essere stata stabilita come LISTNer (consultate LISTN) ed essere passata in un indirizzo secondario se necessario (consultate SECND). Il byte viene passato in .A. Il dato di output seriale viene bufferizzato da un carattere, essendo l'ultimo carattere trasmesso con EOI dopo una chiamata in UNLSN. (La maggior parte delle applicazioni dovrebbero usare le routine di I/O a più alto livello; consultate BSOUT).

15. $FFAB UNTLK ;seriale: invia untalk (non parlare)

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
Memoria:	STATUS ($90)
Flag:	nessuno

ESEMPIO:

```
JSR $FFAB        ;unità seriale UNTALK
```

UNTLK è una routine del bus seriale Kernal a basso livello che invia un comando UNTALK a tutte le unità di bus seriale. Comanda a tutte le unità parlanti (TALKing) di fermare l'invio dei dati. (La maggior parte delle applicazioni dovrebbero usare le routine di I/O a livello più alto; consultate CLRCH).

16. $FFAE UNLSN ;seriale: invia unlisten (non ascoltare)

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
Memoria:	STATUS ($90)
Flag:	nessuno

ESEMPIO:

```
JSR $FFAE        ;unità seriale UNLISTEN
```

UNLSN è una routine del bus seriale Kernal a basso livello che invia un comando UNLISTEN a tutte le unità del bus seriale. Comanda a tutte le unità che ascoltano (LISTENing) di fermare la lettura dei dati. (La maggior parte delle applicazioni dovrebbero usare le routine di I/O a più alto livello; consultate ICLRCH).

17. $FFB1 LISTN ;seriale: invia il comando ascoltare

PREPARAZIONE:

Registri:	.A = unità (0-31)
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
Memoria:	STATUS ($90)
Flag:	nessuno

ESEMPIO:

```
JSR $FFB1        ;comanda all'unità di ascoltare (LISTEN)
```

LISTN è una routine del bus seriale Kernal a basso livello che invia un comando LISTEN all'unità del bus seriale in .A. Comanda all'unità di cominciare a leggere i dati. (La maggior parte delle applicazioni dovrebbero usare le routine di I/O a più alto livello; consultate ICKOUT).

18. $FFB4 TALK ;seriale: invia parlare

PREPARAZIONE:

Registri:	.A = unità (0-31)
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
Memoria:	STATUS ($90)
Flag:	nessuno

ESEMPIO:

```
JSR $FFB4          ;comanda all'unità di parlare (TALK)
```

TALK è una routine del bus seriale Kernal a basso livello che invia un comando TALK all'unità del bus seriale in .A. Comanda all'unità di cominciare ad inviare dati. (La maggior parte delle applicazioni dovrebbero usare routine di I/O a più alto livello; consultate ICHKIN).

19 $FFB7 READSS ;legge l'I/O del byte di stato

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A = STATUS ($90 o $A14)
Memoria:	STATUS cancellato se RS-232
Flag:	nessuno

ESEMPIO:

```
JSR $FFB7          ;STATUS per l'ultimo I/O
```

READSS (alias **READST**) riporta il byte di stato associato all'ultima operazione compiuta di I/O (seriale, cassetta o RS-232). Le operazioni seriale e cassetta a nastro aggiornano lo STATUS ($90) e l'I/O di RS-232 aggiorna RSSTAT ($A14). Notate che per simulare un ACIA, l'RSSTAT è azzerato dopo essere letto via READSS. L'ultima operazione di I/O è determinata dai contenuti di FA ($BA); così le applicazioni che pilotano le unità di I/O usando le chiamate Kernal a basso livello non dovrebbero usare READSS.

20 $FFBA SETLFS ;pone i canali LA, FA, SA

PREPARAZIONE:

Registri:	.A = LA (logico #)
	.X = FA (unità #)
	.Y = SA (indir secondario)
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	nessuno
Memoria:	LA, FA, SA aggiornati
Flag:	nessuno

ESEMPIO:

Consultate OPEN

SETLFS pone il numero del file logico (LA, $B8), il numero di unità (FA, $BA) e l'indirizzo secondario (SA, $B9) per le routine di I/O Kernal a livello più alto. LA deve essere unico tra i file aperti ed è usato per identificare i file specifici per le operazioni di I/O. L'intervallo del numero di unità va da 0 a 31 ed è usato come obiettivo di I/O. SA è un comando che deve essere mandato all'unità indicata, di solito per porla in un modo particolare. Se non si ha bisogno dell'SA, il registro .Y dovrebbe passare in $FF. SETLFS spesso è usato con le chiamate SETNAM e SETBNK prima delle OPEN. Consultate le chiamate Kernal OPEN, LOAD e SAVE per gli esempi.

21. $FFBD SETNAM ;pone i puntatori del nome del file

PREPARAZIONE:

Registri:	.A = lunghezza della stringa
	.X = indir basso della stringa
	.Y = indir alto della stringa
Memoria:	Mappa del sistema
Flag:	nessuno
Chiamate:	SETBNK

RISULTATI:

Registri:	nessuno
Memoria:	FNLEN, FNADR aggiornati
Flag:	nessuno

ESEMPIO:

Consultare OPEN

SETNAM predispone la stringa nome del file o la stringa comando per le chiamate Kernal di I/O ad alto livello come OPEN, LOAD e SAVE. La lunghezza della stringa (nome del file o comando) viene passata in .A ed aggiorna FNLEN ($B7). L'indirizzo della stringa viene passato in .X (basso) e in .Y (alto). Consultate la chiamata compagna, SETBNK, che specifica in quale banco della RAM si trova la stringa. Se non c'è la stringa, si deve chiamare ancora SETNAM e specificare una lunghezza zero ($00) (l'indirizzo non ha importanza). SETNAM viene spesso usato con le chiamate SETBNK e SETLFS prima di quelle OPEN.
Consultate le chiamate Kernal OPEN, LOAD e SAVE per gli esempi.

22. $FFC0 OPEN ;apre il file logico

PREPARAZIONE:

Registri:	nessuno
Memoria:	Mappa del sistema
Flag:	nessuno
Chiamate:	SETLFS, SETNAM, SETBNK

RISULTATI:

Registri:	.A = codice di errore (se ce ne sono) .X = usato .Y = usato
Memoria:	preparazione per I/O STATUS, RSSTAT aggiornati
Flag:	.C = 1 → errore

ESEMPIO: OPEN 1,8 15,"I0"

```
      LDA   #lunghezza          ;fnlen
      LDX   #<nome del file     ;fnadr (comando)
      LDY   #>nome del file
      JSR   $FFBD               ;SETNAM
      LDX   #0                  ;fnbank (Ram 0)
      JSR   $FF68               ;SETBNK
      LDA   #1                  ;la
      LDX   #8                  ;fa
      LDY   #15                 ;sa
      JSR   $FFBA               ;SETLFS
      JSR   $FFC0               ;OPEN
      BCS   errore
nome del file .BYTE"I0"
lunghezza     = 2
```

OPEN prepara un file logico per le operazioni di I/O. Esso crea un'entrata unica nelle tabelle del file logico Kernal LAT ($362), FAT ($36C) e SAT ($376) usando il suo indice LDTND ($98) e dati forniti dall'utente via SETLFS. Ci possono essere fino a dieci file logici aperti (OPENed) simultaneamente. OPEN compie operazioni di apertura specifiche per le unità seriali, cassette ed unità RS-232, incluso l'azzeramento dello stato precedente e la trasmissione di qualsiasi nome di file dato o stringa di comando fornita dall'utente via SETNAM e SETBNK. Lo stato di I/O è aggiornato appropriatamente e può essere letto via READSS.
Il percorso per l'apertura (OPEN) avviene attraverso un vettore indiretto della RAM in $31A. Le applicazioni devono perciò fornire le proprie procedure OPEN o integrare quelle del sistema dirigendo nuovamente questo vettore alle loro routine.

23. $FFC3 CLOSE ;chiude un file logico

PREPARAZIONE:

Registri:	.A = LA (logico #)
Memoria:	mappa del sistema
Flag:	.C (consultate il testo sotto)
Chiamate:	nessuna

RISULTATI:

Registri:	.A = codice di errore (se ce ne sono)
	.X usato
	.Y usato
Memoria:	tabelle logiche aggiornate
	STATUS, RSSTAT aggiornati
Flag:	.C = 1 → errore

ESEMPIO:

```
LDA #1          ;la
JSR $FFC3       ;CLOSE
BCS errore      ;(solo i file del nastro)
```

CLOSE toglie il file logico (LA) passato in .A dalle tabelle dei file logici ed esegue compiti di chiusura specifici per l'unità. Vi passano la tastiera, lo schermo e qualsiasi file non aperto. I file della cassetta aperti per l'output vengono chiusi scrivendo sull'ultimo buffer e (come opzione) un segno EOT. Le unità di I/O RS-232 sono risettate, perdendo qualsiasi dato bufferizzato. I file seriali vengono chiusi trasmettendo un comando CLOSE (se era stato dato un SA quando era aperto), inviando qualsiasi carattere bufferizzato e facendo l'UNLSTN del bus.
C'è uno speciale provvedimento incorporato nella routine CLOSE dei sistemi che hanno come dispositivo un comando DOS del BASIC. Se le condizioni seguenti sono tutte vere, non viene portato a termine un CLOSE completo; la tabella di entrata viene tolta ma un comando CLOSE non viene trasmesso all'unità. Questo fa sì che il canale di comando del dischetto venga aperto (OPENed) e chiuso (CLOSEd) in modo appropriato senza che il sistema operativo del dischetto chiuda tutti i file alla fine:

.C = 1	→ indica un CLOSE speciale
FA >= 8	→ l'unità è un dischetto
SA = 15	→ canale di comando

Il percorso verso CLOSE avviene attraverso un vettore indiretto della RAM in $31C. Le applicazioni possono perciò fornire le proprie procedure di chiusura (CLOSE) o completare quelle del sistema dirigendo nuovamente questo vettore alle loro routine.

24. $FFC6 CHKIN ;pone il canale di input

PREPARAZIONE:

Registri:	.X = LA (logico#)
Memoria:	Mappa del sistema
Flag:	nessuno
Chiamate:	OPEN

RISULTATI:

Registri:	.A = codice di errore (se ce ne sono) .X usato .Y usato
Memoria:	LA, FA, SA, DFLTN STATUS, RSSTAT aggiornati
Flag:	.C = 1 → errore

ESEMPIO:

```
LDX #1          ;la
JSR $FFC6       ;CHKIN
BCS errore
```

CHKIN stabilisce un canale di input verso l'unità associata all'indirizzo logico (LA) passato in .X, in preparazione di una chiamata di BASIN o GETIN. La variabile Kernal DFLTN ($99) viene aggiornata per indicare l'unità di input corrente e le variabili LA, FA ed SA vengono aggiornate con i parametri del file dalla sua entrata nelle tabelle del file logico (messe lì da OPEN). CHKIN compie certe operazioni specifiche per le unità: passano i canali dello schermo e della tastiera, i file della cassetta sono confermati per l'input ed ai canali seriali viene inviato un comando TALK, SA è trasmesso (se necessario). Chiamate CLRCH per ripristinare i normali canali di I/O. CHKIN viene richiesto per tutti gli input tranne che per la tastiera. Se si desidera l'input della tastiera e non viene stabilito nessun altro canale di input, non è necessario chiamare CHKIN o OPEN. La tastiera è l'unità di input per default di BASIN e GETIN.
Il percorso verso CHKIN avviene attraverso un vettore indiretto della RAM in $31E. Le applicazioni devono perciò fornire le proprie procedure CHKIN o integrare quelle del sistema dirigendo nuovamente questo vettore alle loro routine.

25. $FFC9 CKOUT ;pone il canale di output

PREPARAZIONE:

Registri:	.X = LA (logico#)
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	OPEN

RISULTATI:

Registri:	.A = codice di errore (se ce ne sono)
	.X usato
	.Y usato
Memoria:	LA, FA, SA, DFLTO
	STATUS, RSSTAT aggiornati
Flag:	.C = 1 → errore

ESEMPIO:

```
LDX #1          ;la
JSR $FFC9       ;CKOUT
BCS errore
```

CKOUT stabilisce un canale di output verso l'unità associata all'indirizzo logico (LA) passato in .X, in preparazione ad una chiamata di BSOUT. La variabile Kernal DFLTO ($9A) viene aggiornata per indicare l'unità di output corrente e le variabili LA, FA e SA vengono aggiornate con i parametri del file dalla sua entrata nelle tabelle del file logico (messi lì da OPEN). CKOUT compie certe operazioni specifiche per l'unità: i canali della tastiera non sono validi, i canali dello schermo passano, i file della cassetta vengono confermati per l'output ed ai canali seriali viene inviato un comando LISTN e SA viene trasmesso (se necessario). Chiamate CLRCH per ristabilire i normali canali di I/O.
CKOUT viene richiesto per tutti gli output tranne che per lo schermo. Se si desidera l'output dello schermo e non viene stabilito nessun altro canale di output, non c'è bisogno di chiamare CKOUT o OPEN. Lo schermo è l'unità di output per default di BSOUT. Il percorso verso CKOUT avviene attraverso un vettore indiretto della RAM in $320. Le applicazioni possono perciò fornire le loro procedure CKOUT o completare quelle del sistema dirigendo nuovamente questo vettore alle loro routine.

26. $FFCC CLRCH ;ristabilisce i canali di default

PREPARAZIONE:

Registri:	nessuno
Memoria:	Mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
	.X usato
Memoria:	DFLTI, DFLTO aggiornati
Flag:	nessuno

ESEMPIO:

```
JSR $FFCC       ; ristabilisce l'I/O di default
```

CLRCH (alias **CLRCHN**) viene usato per azzerare tutti i canali di apertura e per ristabilire i canali di I/O di default del sistema dopo che altri canali sono stati stabiliti via CHKIN e/o CHKOUT. La tastiera è l'unità di input di default e lo schermo è l'unità di output di default. Se il canale di input fosse verso un'unità seriale, CLRCH prima opera l'UNTLK. Se il canale di output fosse verso un'unità seriale, prima viene operato l'UNLSN.
Il percorso verso CLRCH avviene attraverso un vettore indiretto della RAM in $322. Le applicazioni possono perciò fornire le loro procedure CLRCH o completare quelle del sistema dirigendo nuovamente questo vettore alle loro routine.

27. $FFCF BASIN ;input dal canale

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	CHKIN (se necessario)

RISULTATI:

Registri:	.A = carattere (o codice di errore)
Memoria:	STATUS, RSSTAT aggiornati
Flag:	.C = 1 se errore

ESEMPIO:

```
        LDY #0     ;indice
MORE    JSR $FFCF  ;input di un carattere
        STA dato,Y ;lo bufferizza
INY
        CMP #$0D   ;ritorno del carrello?
        BNE MORE
```

BASIN (alias **CHRIN**) legge un carattere dall'unità corrente di input (DFLTN $99) e lo riporta in .A. Input da unità diverse dalla tastiera (l'unità di input per default) devono subire OPEN e CHKIN. Il carattere viene letto dal buffer di input associato al corrente canale di input:

a. Il dato della cassetta è riportato un carattere per volta dal buffer della cassetta in $B00, leggendo blocchi ulteriori di nastro quando è necessario.
b. Il dato RS-232 è restituito un carattere per volta dal buffer di input dell'RS-232 in $C00, aspettando finchè un carattere viene ricevuto se necessario. Se RSSTAT ($A14) non è corretto da un'operazione precedente, l'input viene saltato e viene sostituito con un input nullo (ritorno del carrello).
c. Il dato seriale è restituito un carattere per volta direttamente dal bus seriale, aspettando finchè un carattere viene inviato se necessario. Se STATUS ($90) non è corretto da un'operazione precedente, l'input viene saltato e viene sostituito con un input nullo (ritorno del carrello).

d. Il dato dello schermo viene letto dalla RAM dello schermo iniziando dalla posizione corrente del cursore e terminando con uno pseudo ritorno del carrello alla fine della linea logica dello schermo. Il modo in cui viene scritta la routine BASIN, la fine della linea (EOL) non è riconosciuta. Gli utenti devono perciò contare da soli i caratteri o altrimenti riconoscere il momento in cui è stato raggiunto EOL logico.
e. Il dato della tastiera viene messo in input accendendo il cursore, leggendo i caratteri dal buffer della tastiera e segnalandoli sullo schermo finchè si incontra un ritorno di carrello. Allora i caratteri vengono restituiti uno per volta dallo schermo finchè tutti i caratteri di input sono passati, incluso il ritorno del carrello. Qualsiasi chiamata dopo EOL farà ricominciare il procedimento.

Il percorso verso BASIN avviene attraverso un vettore indiretto della RAM in $324. Le applicazioni possono perciò fornire le loro procedure BASIN o completare quelle del sistema ridirigendo questo vettore alle loro routine.

28. $FFD2 BSOUT ;output verso il canale

PREPARAZIONE:

Registri:	.A = carattere
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	CKOUT (se necessario)

RISULTATI:

Registri:	.A = codice di errore (se ce ne sono)
Memoria:	STATUS, RSSTAT aggiornato
Flag:	.C = 1 se errore

ESEMPIO:

```
LDA # carattere
JSR $FFD2          ;output di un carattere
```

BSOUT (alias **CHROUT**) scrive il carattere in .A nell'unità di output corrente (DFLTO $9A). Output verso unità diverse dallo schermo (l'unità di output per default) devono subire OPEN e CKOUT. Il carattere viene scritto nel buffer di output associato al canale di output corrente:
a. Il dato della cassetta viene messo un carattere per volta nel buffer della cassetta in $B00, scrivendo sui blocchi del nastro quando è necessario.
b. Il dato dell'RS-232 viene messo un carattere per volta nel buffer di output dell'RS-232 in $D00, aspettando fino a quando c'è spazio se necessario.
c. Il dato seriale viene passato a CIOUT, che bufferizza un carattere ed invia il carattere precedente.
d. Il dato dello schermo viene messo nella RAM dello schermo nella posizione corrente del cursore.
e. L'output della tastiera non è valido.

Il percorso verso BSOUT avviene attraverso un vettore indiretto della RAM in $326. Le applicazioni possono perciò fornire le loro procedure BSOUT o completare quelle del sistema ridirigendo questo vettore alle loro routine.

29. $FFD5 LOAD ;carica dal file

PREPARAZIONE:

Registri:	.A = 0 → LOAD (carica) .A > 0 → VERIFY (verifica) .X = carica indir basso (se SA = 0) .Y = carica indir alto (se SA = 0)
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	SETLFS, SETNAM, SETBNK

RISULTATI:

Registri:	.A = codice di errore (se ce ne sono) .X fine indir basso .Y fine indir alto
Memoria:	secondo comando STATUS aggiornato
Flag:	.C = 1 → errore

ESEMPIO: LOAD "programma",8,1

```
LDA #lunghezza   ;fnlen
LDX #<nome       ;fnadr
del file
LDY #>nome
del file
JSR $FFBD        ;SETNAM
LDA #0           ;carica/verifica il banco (RAM 0)
LDX #0           ;fnbank (RAM 0)
JSR $FF68        ;SETBNK
LDA #0           ;la (non usato)
LDX #8           ;fa
LDY #$FF         ;sa (SA>0 caricamento normale)
JSR $FFBA        ;SETLFS
LDA #0           ;carica, non verifica
LDX #<carica     ;(usato solo se SA = 0)
indir
LDY #>carica     ;(usato solo se SA = 0)
indir
JSR $FFD5        ;LOAD
BCS errore
STX fine basso
STY fine alto
nome del file .BYTE "programma"
lunghezza = 7
```

Questa routine carica (**LOADs**) i dati da un'unità di input nella memoria del C128. Può anche essere usato per VERIFICARE se il dato in memoria corrisponde a quello di un file. LOAD compie delle operazioni specifiche per le unità per LOAD seriali e della cassetta. Non potete caricare dalle unità RS-232, schermo o tastiera. Mentre LOAD compie tutte le operazioni di un OPEN, non crea alcun file logico come fa un OPEN. Notate anche che LOAD non può "avvolgere" banchi di memoria. Come per qualsiasi I/O, lo stato di I/O viene aggiornato in modo appropriato e può essere letto via READSS. LOAD ha due opzioni che l'utente deve selezionare:

a. LOAD contro VERIFY: I contenuti di .A passati alla chiamata verso LOAD determinano in effetti quale modo esso sia. Se .A è zero, sarà compiuta un'operazione LOAD e si scriverà sulla memoria. Se .A non è zero, sarà compiuta un'operazione VERIFY ed il risultato verrà passato via meccanismo dell'errore.
b. LOAD ADDRESS (indirizzo di caricamento): l'indirizzo secondario (SA) predisposto dalla chiamata a SETLFS determina da dove venga l'indirizzo di partenza di LOAD. Se SA è zero, l'utente vuole che venga usato l'indirizzo in .X e .Y al momento della chiamata. Se SA non è zero, l'indirizzo di partenza di LOAD viene letto dal file di testa stesso ed il file viene caricato nello stesso posto dal quale è stato salvato.

La routine seriale LOAD del C128 tenta automaticamente di BURST (interrompere) il caricamento di un file e di fare ricorso al normale meccanismo di caricamento (ma usando ancora le routine seriali VELOCI) se l'handshake del BURST non viene riportato. Il percorso verso LOAD avviene per mezzo di un vettore indiretto della RAM in $330. Le applicazioni possono perciò fornire le loro procedure LOAD o completare quelle del sistema dirigendo nuovamente questo vettore alle loro routine.

30. $FFD8 SAVE ;salva nel file

PREPARAZIONE:

Registri:	.A = puntatore dell'indir di partenza
	.X = fine indir basso
	.Y = fine indir alto
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	SETLFS, SETNAM, SETBNK

RISULTATI:

Registri:	.A = codice di errore (se ce ne sono)
	.X usato
	.Y usato
Memoria:	STATUS aggiornato
Flag:	.C = 1 → errore

ESEMPIO: SAVE "programma",8

```
LDA #lunghezza   ;fnlen
LDX #<nome       ;fnadr
del file
```

```
    LDY #>nome
    del file
    JSR $FFBD          ;SETNAM
    LDA #0             ;salva dal banco (RAM 0)
    LDX #0             ;fnbank (RAM 0)
    JSR $FF68          ;SETBNK
    LDA #0             ;la (non usato)
    LDX #8             ;fa
    LDY #0             ;sa (solo cassetta)
    JSR $FFBA          ;SETLFS
    LDA ©start         ;puntatore all'indirizzo di partenza
    LDX end            ;fine indirizzo basso
    LDY end + 1        ;fine indirizzo alto
    JSR $FFD8          ;SAVE
    BCS errore
nome del file .BYTE "programma"
lunghezza     = 7
inizio        .WORD indirizzo 1; pag-0
fine          .WORD indirizzo 2
```

Questa routine SALVA (**SAVE**) il dato dalla memoria del C128 verso un'unità di output. SAVE compie delle operazioni specifiche per l'unità per salvataggi seriali e su cassetta. Non potete salvare da unità RS-232, schermo o tastiera. Mentre SAVE esegue tutti i compiti di un OPEN, non crea file logici come fa un OPEN.
L'indirizzo di partenza dell'area che deve essere salvata deve essere posto in un vettore di pagina zero e l'indirizzo di questo vettore deve essere passato a SAVE in .A al momento della chiamata. L'indirizzo dell'ultimo byte che deve essere salvato PIÙ UNO viene passato in .X e .Y allo stesso tempo. I salvataggi della cassetta utilizzano l'indirizzo secondario (SA) per specificare il tipo di testata(e) del nastro che devono essere generate:

SA (bit 0) = 0 → file rilocabile (blf)
           = 1 → file assoluto (plf)
SA (bit 1) = 0 → fine normale
           = 1 → scrive la testata EOT alla fine

Non c'è salvataggio di BURST; vengono usate le normali routine seriali FAST (veloci). Come per ogni I/O, l'I/O di stato sarà aggiornato in modo appropriato e può essere letto via READSS.
Il percorso per SAVE avviene per mezzo di un vettore indiretto della RAM in $332. Le applicazioni possono perciò fornire le loro procedure SAVE o completare quelle del sistema dirigendo nuovamente questo vettore alle loro routine.

31. $FFDB SETTIM ;pone un clock interno

PREPARAZIONE:

Registri:	.A = byte basso .X = byte medio .Y = byte alto
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	nessuno
Memoria:	TIME ($A0) aggiornato
Flag:	nessuno

ESEMPIO:

```
LDA #0        ;risetta il clock
TAX
TAY
JSR $FFDB     ;SETTIM
```

SETTIM pone il clock software (jiffie) del sistema, che conta i sessantesimi (1/60) di secondo. Il timer viene incrementato durante il procedimento del sistema IRQ (consultate UDTIM) e viene risettato alla 24esima ora. SETTIM disabilita gli IRQ, aggiorna il timer a tre byte con i contenuti di .A, .X e .Y, e riabilita gli IRQ.

32. $FFDE RDTIM ;legge il clock interno

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A = byte basso .X = byte medio .Y = byte alto
Memoria:	nessuna
Flag:	nessuno

ESEMPIO:

```
JSR $FFDE     ;RDTIM
```

RDTIM legge il clock software (jiffie) del sistema, che conta i sessantesimi (1/60) di secondo. Il timer viene incrementato durante il procedimento del sistema IRQ (consultate UDTIM), e risettato alla 24esima ora. RDTIM disabilita gli IRQ, carica .A, .X e.Y con i contenuti del timer a 3 byte e riabilita gli IRQ.

33. $FFE1 STOP ;localizza il tasto STOP

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A = ultima riga della tastiera
	.X = usato (se tasto STOP)
Memoria:	nessuna
Flag:	stato valido

ESEMPIO:

```
JSR $FFE1        ;localizza il tasto STOP
BEQ stop         ;salta se abbassato
```

STOP controlla una variabile Kernal STKEY ($91), che viene aggiornata da UDTIM durante il normale procedimento IRQ e contiene l'ultima localizzazione della colonna C7 della tastiera. Il tasto STOP è il bit 7, che sarà 0 se il tasto è abbassato. Se lo è, i canali di default di I/O vengono ristabiliti via CLRCH e la coda della tastiera viene pulita risettando NDX ($D0). I tasti sulla linea C7 della tastiera sono:

Bit:	7	6	5	4	3	2	1	0
Tasto:	STOP	Q	C=	SPAZIO	2	CTRL	←	1

Il percorso verso STOP avviene per mezzo di un vettore indiretto della RAM in $328. Le applicazioni possono perciò fornire le loro procedure STOP o completare quelle del sistema dirigendo nuovamente questo vettore alle loro routine.

34. $FFE4 GETIN ;legge il dato bufferizzato

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	CHKIN (se necessaria)

RISULTATI:

Registri:	.A = carattere (o codice di errore)
	.X usato
	.Y usato
Memoria:	STATUS, RSSTAT aggiornati
Flag:	.C = 1 se errore

ESEMPIO:

```
aspetta JSR $FFE4      ;ottiene qualsiasi tasto
        BEQ aspetta
        STA carattere
```

GETIN legge un carattere dal buffer dell'unità di input corrente (DFLNT ($99)) e la riporta in .A. Input da unità diverse dalla tastiera (l'unità di input per default) devono subire OPEN e CHKIN. Il carattere viene letto dal buffer di input associato al canale di input corrente:

a. Input della tastiera: Un carattere viene rimosso dal buffer dellA tastiera e passato in .A. Se il buffer è vuoto, viene restituito uno zero ($00).
b. Input dell'RS-232: Un carattere viene rimosso dal buffer di input dell'RS-232 in $C00 e passato in .A. Se il buffer è vuoto, viene restituito uno zero ($00) (usate READSS per controllare la validità).
c. Input seriale: GETIN salta automaticamente a BASIN. Consultate l'I/O seriale BASIN.
d. Input della cassetta: GETIN salta automaticamente a BASIN. Consultate l'I/O della cassetta BASIN.
e. Input dello schermo: GETIN salta automaticamente a BASIN. Consultate l'I/O seriale BASIN.

Il percorso verso GETIN avviene per mezzo di un vettore indiretto della RAM in $32A. Le applicazioni possono perciò fornire le loro procedure GETIN o completare quelle del sistema ridirigendo questo vettore alle loro routine.

35. $FFE7 CCALL; chiude tutti i file ed i canali

PREPARAZIONE

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato .X usato
Memoria:	LDTND, DFLTN, DFLTO aggiornati
Flag:	nessuno

ESEMPIO:

```
JSR $FFE7      ;chiude i file
```

CLALL cancella tutte le entrate della tabella dei file logici risettando l'indice della tabella, LDTND ($98). Essa cancella i canali seriali correnti (se ce ne sono) e ripristina i canali di I/O per default via CLRCH.
Il percorso verso CALL avviene attraverso un vettore indiretto della RAM in $32C. Le applicazioni possono perciò fornire le loro procedure CLALL o

completare quelle del sistema dirigendo nuovamente questo vettore alle loro routine.

36. $FFEA UDTIM; incrementa il clock interno

PREPARAZIONE

Registri:	nessuno
Memoria:	mappa del sistema
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
	.X usato
Memoria:	TIME, TIMER, STKEY aggiornati
Flag:	nessuno

ESEMPIO:

```
SEI
JSR $FFEA        ;UDTIM
CLI
```

UDTIM incrementa il clock di software (jiffie) del sistema, che conta i sessantesimi (1/60) di secondo quando è chiamato dall'IRQ 60Hz del sistema. TIME, un contatore a 3 byte localizzato in $A0, viene risettato alla 24esima ora, UDTIM decrementa anche TIMER, pure un contatore a 3 byte, localizzato in $A1D (il BASIC lo usa per il comando SLEEP, per esempio). Dovreste essere sicuri del sistema verso UDTIM mentre state modificando TIME e TIMER.
UDTIM localizza anche la linea C7 del tasto sulla quale c'è il tasto STOP, e memorizza il risultato in STKEY ($91). La routine Kernal STOP utilizza questa variabile.

37. $FFED SCRORG ;ottiene la finestra corrente dello schermo

PREPARAZIONE:

Rergistri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI

Registri:	.A = larghezza dello schermo
	.X = larghezza della finestra
	.Y = altezza della finestra

ESEMPIO

```
JSR $FFED        ;SCRORG
```

SCRORG è una routine dell'Editor che è stata leggermente cambiata dai sistemi CBM precedenti. Invece di riportare le dimensioni massime dello SCHERMO in .X e .Y, lo SCRORG del C128 riporta le dimensioni della finestra (WINDOW) corrente. Esso riporta la massima larghezza dello SCHERMO in .A. Queste modifiche fanno sì che le applicazioni "si adattino" alla finestra corrente dello schermo. SCRORG è anche un'entrata della tabella di salto dell'Editor ($C00F).

38. $FFF0 PLOT; legge/pone la posizione del cursore

```
PREPARAZIONE
    Registri:          .X = linea del cursore
                       .Y = colonna del cursore
    Memoria:           mappa del sistema
    Flag:              : .C = 0 → pone la posizione del cursore
                         .C = 1 → ottiene la posizione del cursore
    Chiamate:          nessuna

RISULTATI:
    Registri:          .X = linea del cursore
                       .Y = colonna del cursore
    Memoria:           TBLX, PNTR aggiornati
    Flag:              .C = 1 → errore
```

ESEMPIO:

```
SEC
JSR $FFF0        ; legge la posizione corrente
INX              ; abbassa di una linea
INY              ; muove a destra di uno spazio
CLC
JSR $FFF0        ; pone la posizione del cursore
BCS errore       ; nuova posizione fuori della finestra
```

PLOT è una routine dell'Editor che è stata leggermente modificata dai precedenti sistemi CBM. Invece di utilizzare delle coordinate *assolute* quando fa riferimento alla posizione del cursore, PLOT ora utilizza delle coordinate *relative,* basate sulla *finestra* corrente dello schermo. Le seguenti variabili *locali* dell'Editor sono utili:

```
SCBOT       $E4 → basso della finestra
SCTOP       $E5 → alto della finestra
SCLF        $E6 → lato sinistro della finestra
SCRT        $E7 → lato destro della finestra
TBLX        $EB → linea del cursore
PNTR        $EC → colonna del cursore
LINEE       $ED → massima altezza dello schermo
COLONNE     $EE → massima larghezza dello schermo
```

Quando è chiamato con il riporto di stato, PLOT dà la posizione del cursore corrente relativa all'origine della finestra corrente (*non* all'origine dello schermo). Quando è chiamato con il cancellamento del riporto di stato, PLOT tenta di muovere il cursore sulle linea e colonna indicate relative all'origine della finestra corrente (*non* all'origine dello schermo). PLOT riporterà un cancellamento del riporto di stato se il cursore era spostato ed un posizionamento del riporto di stato se la posizione richiesta era fuori dalla finestra corrente (*non è stato fatto alcun cambiamento*).

39. $FFF3 IOBASE ;legge l'indirizzo base di I/O del blocco

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.X = lsb dell'I/O del blocco
	.Y = msb dell'I/O del blocco
Memoria:	nessuna
Flag:	nessuno

ESEMPIO:

```
JSR $FFF3          ; trova l'I/O del blocco
```

IOBASE non è usato nel C128 ma viene incluso per compatibilità e completezza. Riporta l'indirizzo del blocco di I/O in .X e .Y.

NUOVE CHIAMATE KERNAL DEL C128

Le seguenti chiamate del sistema sono un insieme di estensioni della tabella di salto CBM standard. Esse sono specificatamente per il C128 e come tali non dovrebbero essere considerate come aggiunte permanenti alla tabella di salto standard. Con l'eccezione del MODO C64, esse sono tutte vere subroutine e termineranno per mezzo di un RTS. Come per tutte le chiamate Kernal, la configurazione del sistema (ROM alta, RAM-0 e I/O) devono essere in contesto al momento della chiamata.

1. $FF47 SPIN SPOUT ;predispone gli ingressi seriali per l'I/O

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	.C = 0 → seleziona SPINP
	.C = 1 → seleziona SPOUT
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
Memoria:	CIA-1, MMU
Flag:	nessuno

ESEMPIO:

```
CLC
JSR $FF47           ; predispone un input seriale veloce
```

Il protocollo seriale veloce del C128/1571 utilizza il CIA 1 (6526 in $DC00) ed uno speciale circuito driver controllato in parte dall'MMU (in $D500). **SPINP** e **SPOUT** sono routine utilizzate dal sistema per predisporre il CIA ed il circuito driver seriale veloce per l'input o l'output. SPINP predispone il CRA (CIA 1 registro 14) ed azzera il bit FSDIR (MMU registro 5) per l'input. SPOUT predispone CRA, ICR (CIA 1 registro 13), il timer A (CIA 1 registri 4 e 5) e pone il bit FSDIR per l'output. Notate che lo stato del bit TODIN di CRA è sempre mantenuto, ma lo stato degli output GAME, EXROM e SENSE40 di MMU non lo sono (la lettura di questi ingressi riporta lo stato dell'ingresso e non i valori dei registri- di conseguenza non possono essere mantenuti). Queste routine vengono richieste solo da applicazioni che pilotano da sole il bus seriale veloce dal livello più basso.

2. $FF4A CLOSE ALL ;chiude tutti i file di un'unità

PREPARAZIONE:

Registri:	.A → unità # (FA:0-31)
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
	.X usato
	.Y usato
Memoria:	nessuna
Flag:	nessuno

ESEMPIO:

```
LDA #$08
JSR $FF4A          ;chiude tutti i file dell'unità 8
```

FAT è richiesto per FA stabilito. Viene compiuta una chiusura (CLOSE) appropriata per tutte le corrispondenze. Se uno dei canali chiusi è il canale corrente di I/O, allora viene ripristinato il canale di default. Questa chiamata viene utilizzata, per esempio, dal comando **DCLOSE** del BASIC. Viene anche chiamata dalla routine BOOT del Kernal.

3. $FF4D C64MODE ;riconfigura il sistema come un C64

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	nessuno
Memoria:	nessuna
Flag:	nessuno

ESEMPIO:

```
JMP $FF4D          ;salta al modo C64
```

Non c'è ritorno da questa routine. Il DDR 8502 e l'ingresso vengono inizializzati ed il VIC è posto in modo 1MHz (lento). Il controllo viene passato al codice nella RAM comune (condivisa), che pone il registro del modo MMU (#5) in modo C64. Da questo punto in avanti, le ROM MMU e C128 non sono accessibili. La routine esce con un salto indiretto attraverso il vettore RESET del C64.
Poichè l'operazione del C64 non permette l'accesso all'MMU, tutti i registri MMU devono essere configurati per operazioni appropriate prima che venga posto il bit del modo C64. In modo simile, poichè la partenza del sistema operativo del C64 non deriva da un vero reset dell'hardware, c'è la possibilità che stati insoliti di I/O che sono in effetti precedenti a chiamate C64MODE possano causare situazioni non prevedibili e presumibilmente non desiderabili una volta che si è in modo C64.
Non c'è possibilità di tornare dal modo C64 al modo C128; solo un reset dell'hardware o spegnimento/accensione possono ristabilire il modo C128 dell'operazione. Un reset inizierà sempre il modo C128, sebbene l'alterazione anticipata del vettore SYSTEM sia un modo per "gettare" automaticamente un sistema in modo C64.

4. $FF50 DMA CALL ;invia il comando all'unità DMA

PREPARAZIONE:

Registri:	.X = banco (0-15)
	.Y = unità di controllo DMA del comando
Memoria:	registri DMA che predispongono la mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
	.X usato
Memoria:	modificata secondo il comando
Flag:	nessuno

ESEMPIO:

```
LDA #$00     ;predispone l'indirizzo di base del C128
STA $DF02    ;basso
LDA #$20
STA $DF03    ;alto
LDA #$00     ;predispone l'espansione dell'indirizzo della RAM
STA $DF04    ;basso
STA $DF05    ;alto
STA $DF06    ;banco (0-n, dove n=3 se 256K)
LDA #$40     ;predispone il numero di byte
STA $DF07    ;basso
LDA #$1F
STA $DF08    ;alto
LDX #$00     ;banco del C128
LDY #$84     ;comando DMA verso "STASH"
JSR $FF50    ;esegue il comando DMA
```

DMA CALL (chiamata) è progettata per comunicare con una cartuccia ad espansione esterna aperta a DMA ed inserita nella memoria del sistema in IO2 ($DFxx). La chiamata DMA (DMA CALL) converte il parametro logico del banco del C128 in una configurazione MMU via GETCFG, gli OR nel bit abilitatore di I/O e trasferisce il controllo al codice della RAM in $3F0. Qui il banco specificato del C128 viene portato nel contesto ed il comando dell'utente viene inviato all'unità di controllo del DMA. Il trasferimento reale del DMA viene eseguito a questo punto, tenendo l'8502 lontano dal bus in uno stato di attesa. Non appena l'unità di controllo del DMA libera il processore, la memoria viene riconfigurata nello stato in cui era al momento della chiamata ed il controllo viene ridato a chi chiama. L'utente deve analizzare lo stato di completamento leggendo il registro di stato del DMA in $DF00.
Si deve fare attenzione nell'utilizzare il prodotto di espansione della RAM del C128 con qualsiasi applicazione usando l'interfaccia incorporata del Kernal. Questa include specialmente l'uso dei comandi in BASIC del C128

FETCH, STASH e SWAP. Nella routine che prepara una richiesta DMA per l'utente, il Kernal forza il blocco di I/O perchè sia sempre in contesto. Di conseguenza, è probabile che il dato che viene dall'unità DMA rovini le unità di I/O sensibili. Gli utenti dovrebbero o oltrepassare la routine Kernal DMA fornendo la propria interfaccia, o limitare i trasferimenti dei dati DMA alle aree sopra e sotto il blocco di I/O. Solo un controllo severo di quest'ultima garantirà un'utilizzazione adeguata dei comandi in BASIC. Il codice seguente, usato al posto della DMA CALL nell'esempio precedente, illustra una procedura:

```
LDX #$00        ;banco del C128
LDY #$84        ;comando DMA verso "STASH"
JSR $FF6B       ;GETCFG
TAX
JSR $3F0        ;esegue un comando DMA
```

5. $FF53 BOOT CALL ;fa il boot d/ programma di caricamento dal dischetto

PREPARAZIONE:

Registri:	.A = numero del drive (ASCII)
	.X = numero dell'unità (0-31)
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
	.X usato
	.Y usato
Memoria:	cambiata secondo il programma
Flag:	.C → 1 se errore di I/O

ESEMPIO:

```
LDA #$30        ;drive 0
LDX #$08        ;unità 8
JSR $FF53       ;BOOT
BCS IO ERROR
BCC NO BOOT SECTOR
```

BOOT tenta di caricare ed eseguire il settore del boot (boot sector) da un dischetto di auto-boot in drive ed unità stabiliti. Il protocollo del BOOT è il seguente:

a. Chiude tutti i file aperti dell'unità di boot.
b. Legge la traccia 1 settore 0 in TBUFFR ($B00).

c. Controlla l'auto-boot di testa, se non RTS.
d. Se (blk# > 0), il BLOCCO LEGGE (BLOCK READ) i settori sequenziali nella RAM nelle locazioni date (adrl, adrh, banco).
e. Se LEN (nome del file) > 0, CARICA il file nella RAM-0 (caricamento normale).
f. JSR verso il codice dell'utente nella locazione C sopra.

Con qualsiasi errore, l'operazione di BOOT viene fermata e viene inviato il comando IU al dischetto. Un ritorno può o non può essere fatto a chi chiama, ciò dipende dallo stato di completamento e dal codice di cui si è fatto il boot. Il settore di BOOT ha il seguente tracciato:

$00	$01	$02	$03	$04	$05	$06		A		B	C
C	B	M	adrl	adrh	bank	blk#	title	0	file	0	code

dove: A = $07 + LEN(titolo)
B = A + LEN(nome del file)
C = B + 1

Gli esempi seguenti illustrano la flessibilità di questo tracciato. Esso carica ed esegue un programma in BASIC:

```
$00  → CBM                 :tasto
$03  → $00,$00,$00,$00     :no altri settori di BOOT
$07  → NAME,$00            :messaggio "NOME"
$0C  → $00                 :no nome del file
$0D  → $A2,$13,$A0,$0B     :codice
       $4C,$A5,$AF
$14  → RUN "PROGRAM"       :dato(BASIC stmt)
$20  → $00
```

Questo risulta nel messaggio **Booting NAME** ... che viene visualizzato e, utilizzando un'entrata della tabella di salto in BASIC del C128 che trova ed esegue un'istruzione in BASIC, carica ed esegue il programma in BASIC chiamato "PROGRAM". Lo stesso header (testata) può essere usato per caricare ed eseguire un programma binario (in codice macchina) cambiando semplicemente RUN in BOOT. (Mentre la funzione auto caricante del file dell'header del boot potrebbe essere usata per caricare file binari semplicemente fornendo un nome di file, per eseguirlo dovete conoscere l'indirizzo di partenza e JMP ad esso. Il comando di BOOT in BASIC lo fa, e permette un meccanismo più generico). Nell'esempio seguente, viene visualizzato un menu e vi si domanda di selezionare il modo operativo. Non è caricato nient'altro in questo header di tipo "configurato":

```
$00  → CBM                      :tasto
$03  → $00,$00,$00,$00          :no altro settore del BOOT
$07  → $00                      :no messaggio
$0C  → $00                      :no nome del file
$0D  → $20,$7D,$FF,$0D,$53,$45,$4C,$45
       $43,$54,$20,$4D,$4F,$44,$45,$3A
       $0D,$0D,$20,$31,$2E,$20,$43,436
       $34,$20,$20,$42,$41,$53,$49,$43
       $0D,$20,$32,$2E,$20,$43,$31,$32
       $38,$20,$42,$41,$53,$49,$43,$0D
       $20,$33,$2E,$20,$43,$31,$32,$38
       $20,$4D,$4F,$4E,$49,$54,$4F,$52
       $0D,40D,$00,$20,$E4,$FF,$C9,$31
       $D0,$03,$4C,$4D,$FF,$C9,$32,$D0
       $03,$4C,$03,$40,$C9,$33,$D0,$E3
           $4C,$00,$B0
```

Il caricamento dei settori sequenziali è progettato prima di tutto per le applicazioni specializzate (come CP/M o i giochi) che non hanno bisogno di un'entrata nel direttorio del dischetto.

6. $FF56 PHOENIX ;inizializza le cartucce di funzione

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa della cartuccia
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
	.X usato
	.Y usato
Memoria:	cambiata secondo il comando
Flag:	nessuno

ESEMPIO:

```
JSR $FF56        ;PHOENIX
```

La routine Kernal POLL di inizializzazione del C128 crea una Tabella di Indirizzo Fisico **(Phisical Address Table, PAT)** che contiene l'ID di tutte le cartucce di funzione ROM installate. PHOENIX chiama ogni entrata registrata a partenza a freddo della cartuccia nell'ordine: esterno basso/alto ed interno basso/alto. Dopo aver chiamato le cartucce (se ce ne sono), PHOENIX chiama la routine Kernal BOOT perchè cerchi un dischetto auto-boot nel drive 0 dell'unità 8 (consultate BOOT CALL). Il controllo può essere o non essere riportato all'utente. PHOENIX viene chiamato dal BASIC alla fine della sua inizializzazione a freddo.

7. $FF59 LKUPLA ;cerca le tabelle per LA stabilito

8. $FF5C LKUPSA ;cerca le tabelle per SA stabilito

PREPARAZIONE:

Registri:	.A = LA (numero del file logico) se LKUPLA .Y = SA (indirizzo secondario) se LKUPSA
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A = LA (solo se trovato) .X = FA (solo se trovato) .Y = SA (solo se trovato)
Memoria:	nessuna
Flag:	.C = 0 se trovato .C = 1 se non trovato

ESEMPIO:

```
LDY #$60          ;trova un SA disponibile
AGAIN INY
CPY #$6F
BCS TOO MANY      ;troppi file aperti
JSR $FF5C         ;LKUPSA
BCC AGAIN         ;ottiene un altro se in uso
```

LKUPLA e **LKUPSA** sono routine Kernal usate in primo luogo da comandi DOS del BASIC per operare su canali del dischetto aperti dall'utente. Il Kernal richiede unici numeri logici di unità (LA) ed il dischetto richiede indirizzi secondari unici (SA); perciò il BASIC deve trovare valori alternativi non usati tutte le volte che ha bisogno di stabilire un canale del dischetto.

9. $FF5F SWAPPER ;si muove tra 40 e 80 colonne

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato .X usato .Y usato
Memoria:	variabili locali scambiate
Flag:	nessuno

ESEMPIO:

LDA $D7:	verifica il modo video
BMI is—80:	salta se di 80 colonne
JSR $FF5F:	si muove da 40 ad 80

SWAPPER è un'utility dell'Editor usata per passare tra il video VIC a 40 colonne (composto) ed il video dell'8563 (RGBI) ad 80 colonne. La routine scambia semplicemente le variabili locali (associate ad uno schermo particolare), le tabelle TAB e le mappe di linee scandite (line wrap maps) con quelle che descrivono l'altro schermo. L'MSB del MODE, in locazione $D7, è fissato da SWAPPER per indicare il modo di visualizzazione corrente: $80 = 80 colonne, $00 = 40 colonne.

10. $FF62 DLCHR ;inizializza la RAM ad 80 colonne

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato .X usato .Y usato
Memoria:	RAM dell'8563 inizializzata
Flag:	nessuno

ESEMPIO:

JSR $FF62	;inizializza le definiz del carattere dell'8563

DLCHR (alias INIT80) è un'utility dell'Editor per copiare le definizioni dei caratteri del VIC dalla ROM ($D000—$DFFF, banco 14) alla RAM video dell'8563 ($2000—$3FFF, locale all'8563 — non nello spazio dell'indirizzo del processore). Le celle del carattere 8 per 8 del VIC vengono normalizzate con degli zeri ($00) per riempire le celle dei caratteri 8 per 16 dell'8563. Consultate il Capitolo 11, la Programmazione del Chip ad 80 Colonne (8563) per dettagli concernenti la fonte del tracciato dell'8563.

11. $FF65 PFKEY ;programma un tasto di funzione

PREPARAZIONE:

Registri:	.A = puntatore all'indir. stringa (basso/alto/banco) .Y = lunghezza della stringa .X = numero del tasto (1—10)
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A usato
	.X usato
	.Y usato
Memoria:	PKYBUF, PKYDEF tabelle aggiornate
Flag:	.C = 0 se riuscito
	.C = 1 se non c'è spazio disponibile

ESEMPIO:

```
LDA #$FA          ;puntatore dell'indirizzo della stringa
LDY #$06          ;lunghezza
LDX #$0A          ;tasto# (tasto HELP)
JSR $FF65         ;installa le nuove definizioni dei tasti
BCS NO ROOM

>000FA 00 13 00            :ptr in $1300 banco 0
>01300 53 54 52 49 4E 47 :"stringa"
```

PFKEY (alias KEYSET) è un'utility dell'Editor che sostituisce una stringa di un tasto di funzione del C128 con una stringa dell'utente. I tasti 1–8 sono F1–F8, 9 è la stringa RUN **SHIFT** e 10 è la stringa HELP. L'esempio precedente sostituisce la stringa di "aiuto" **RETURN** assegnata all'inizializzazione del sistema al tasto **HELP** con la stringa "stringa". Sia la tabella di lunghezza dei tasti, PKYBUF ($1000–$1009) che l'area di definizione, PKYDEF ($100A–$10FF) vengono condensate ed aggiornate. La lunghezza massima di tutte le 10 stringhe è di 246 caratteri. Non si fanno modifiche se non c'è spazio sufficiente per una nuova definizione.

12. $FF68 SETBNK ;pone il banco per le operazioni di I/O

PREPARAZIONE:

Registri:	.A = BA, banco di memoria (0–15)
	.X = FNBANK, banco del nome del file
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	SETNAM

RISULTATI:

Registri:	nessuno
Memoria:	BA, FNBANK aggiornati
Flag:	nessuno

ESEMPIO:

Consultate OPEN

SETBNK è un presupposto per qualsiasi operazione di I/O della memoria e deve essere utilizzato con SETLFS e SETNAM prima di aprire i file, etc. BA

($C6) pone il banco corrente di memoria 64KB per le operazioni LOAD/SAVE/VERIFY. FNBANK ($C7) indica il banco in cui si trova la stringa del nome del file. La routine Kernal GETCFG viene usata per tradurre i numeri logici stabiliti del banco (0—15). SETBNK spesso è usato con le chiamate SETNAM e SETLFS prima delle OPEN. Consultate le chiamate Kernal OPEN, LOAD e SAVE per gli esempi.

13. $FF6B GETCFG ;cerca il dato MMU per il banco stabilito

PREPARAZIONE:

Registri:	.X = banco logico# (0—15)
Memoria:	mappa del sistema
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.A = configurazione del dato MMU
Memoria:	nessuna
Flag:	nessuno

ESEMPIO:

```
LDX #$00;       banco logico 0 (RAM O)
JSR $FF6B;      GETCFG
STA $FF01;      predispone la preconfigurazione # di MMU
```

GETCFG permette un approccio universale, logico ai numeri del banco fisico fornendo una semplice conversione di ricerca per ottenere l'attuale configurazione di dati MMU. In tutti i casi in cui si richiede un numero di banco 0-15, vi potete aspettare che GETCFG sia chiamato per convertire quel numero di conseguenza. Non c'è errore di verifica; se il numero logico del banco stabilito è fuori intervallo il risultato non è valido. Fate riferimento all'Unità di Gestione della Memoria del Commodore 128 più avanti in questo capitolo per avere dettagli sulla configurazione della memoria. I banchi Kernal di memoria del C128 vengono assegnati come segue:

```
 0. %00111111 :solo RAM 0
 1. %01111111 :solo RAM 1
 2. %10111111 :solo RAM 2
 3. %11111111 :solo RAM 3
 4. %00010110 :INT ROM, RAM 0, I/O
 5. %01010110 :INT ROM, RAM 1, I/O
 6. %10010110 :INT ROM, RAM 2, I/O
 7. %11010110 :INT ROM, RAM 3, I/O
 8. %00101010 :EXT ROM, RAM 0, I/O
 9. %01101010 :EXT ROM, RAM 1, I/O
10. %10101010 :EXT ROM, RAM 2, I/O
11. %11101010 :EXT ROM, RAM 3, I/O
12. %00000110 :KERNAL, INT LO, RAM 0, I/O
13. %00001010 :KERNAL, EXT LO, RAM 0, I/O
14. %00000001 :KERNAL, BASIC, RAM 0, CHAR ROM
15. %00000000 :KERNAL, BASIC, RAM 0, I/O
```

14. $FF6E JSRFAR ;gosub in un altro banco

15. $FF71 JMPFAR ;goto un altro banco

PREPARAZIONE:

Registri:	nessuno
Memoria:	mappa del sistema, anche:
	$02 → banco (0−15)
	$03 → PC alto
	$04 → PC basso
	$05 → .S (stato)
	$06 → .A
	$07 → .X
	$08 → .Y
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	nessuno
Memoria:	come per la chiamata, anche:
	$05 → .S (stato)
	$06 → .A
	$07 → .X
	$08 → .Y
Flag:	nessuno

Le due routine, **JSRFAR** e **JMPFAR,** abilitano l'esecuzione del codice nel banco del sistema della memoria per chiamare (o JMP in) una routine in qualsiasi altro banco. Nel caso di JSRFAR, si farà ritorno al banco di chi chiama. Si dovrebbe notare che JSRFAR chiama JMPFAR, che chiama GETCFG. Quando chiama un banco non del sistema, l'utente dovrebbe prendere le precauzioni necessarie per assicurarsi che le interruzioni (IRQ e NMI) saranno gestite in modo appropriato (o disabilitate prima). Sia JSRFAR che JMPFAR sono routine basate sulla RAM e localizzate nella RAM comune (condivisa) in $2CD e $2E3 rispettivamente. Il codice seguente illustra come chiamare una subroutine nel secondo banco della RAM dal banco del sistema. Non ci preoccupiamo di IRQ e NMI in questo caso perchè il sistema li gestirà in modo adeguato in qualsiasi configurazione che abbia la ROM Kernal o qualsiasi banco RAM valido nel contesto nella pagina più alta della memoria.

```
STY $08      ;assume i registri e lo stato
STX $07      ;predispone già per la chiamata
STA $06
PHP
PLA
STA $05
LDA #1       ;vuole chiamare $2000 nel banco 1
LDY #$20
LDX #$00
STA $02
```

```
STY $03
STX $04
JSR $FF6E    ;JSRFAR
LDA $05      ;ripristina lo stato ed i registri
PHA
LDA $06
LDX $07
LDY $08
PLP
```

16. $FF74 INDFET ;LDA (fetvec), Y da qualsiasi banco

```
PREPARAZIONE:
    Registri:      .A = puntatore dell'indirizzo
                   .X = banco (0-15)
                   .Y = indice
    Memoria:       predispone il vettore indiretto
    Flag:          nessuno
    Chiamate:      nessuna

RISULTATI:
    Registri:      .A = dato
                   .X usato
    Memoria:       nessuna
    Flag:          stato valido
```

ESEMPIO:

```
LDA #$00       ;predispone alla lettura di $2000
STA $FA
LDA #$20
STA $FB
LDA #$FA
LDX #$01       ;nel banco 1
LDY #$00
JSR $FF74      ;LDA ($FA, RAM 1), Y
BEQ etc
```

INDFET abilita le applicazioni perchè leggano i dati da qualsiasi altro banco. Esso predispone FETVEC ($2AA), chiama GETCFG per convertire il numero del banco e JMP al codice nella RAM comune (condivisa) in $2A2 che commuta i banchi, carica i dati, ristabilisce il banco dell'utente e torna al programma principale. Quando chiama un banco non di sistema, l'utente dovrebbe prendere le precauzioni necessarie per assicurarsi che le interruzioni (IRQ e NMI) saranno trattate in modo appropriato (o disabilitate prima).

17. $FF77 INDSTA ;STA (stavec), Y in qualsiasi banco

PREPARAZIONE:

Registri:	.A = dato .X = banco (0−15) .Y = indice
Memoria:	predispone il vettore indiretto predispone il puntatore STAVEC ($2B9)
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.X usato
Memoria:	cambiata per chiamata
Flag:	stato non valido

ESEMPIO:

```
LDA #$00        ;predispone la scrittura in $2000
STA $FA
LDA #$20
STA $FB
LDA #$FA
STA $2B9
LDA dato
LDX #$01        ;nel banco 1
LDY #$00
JSR $FF77       ;STA ($FA ,RAM 1), Y
```

INDSTA abilita le applicazioni per la scrittura dei dati in qualsiasi altro banco. Dopo aver predisposto STAVEC ($2B9), chiama GETCFG per convertire il numero del banco e JMP al codice nella RAM comune (condivisa) in $2AF che commuta i banchi, memorizza i dati, ripristina il vostro banco e ritorna al programma principale. Quando chiama un banco non di sistema, l'utente dovrebbe prendere le precauzioni necessarie per assicurarsi che le interruzioni (IRQ e NMI) saranno trattate in modo adeguato (o disabilitate prima).

18. $FF7A INDCMP ;CMP (cmpvec), Y in qualsiasi banco

PREPARAZIONE:

Registri:	.A = dato .X = banco (0−15) .Y = indice
Memoria:	predispone il vettore indiretto predispone il puntatore di CMPVEC ($2C8)
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	.X usato
Memoria:	nessuna
Flag:	stato valido

ESEMPIO:

```
LDA #$00        ;predispone la verifica $2000
STA $FA
LDA #$20
STA $FB
LDA #$FA
STA $2C3
LDA dato
LDX #$01        ;nel banco 1
LDY #$00
JSR $FF7A       ;CMP ($FA, RAM 1), Y
BEQ lo stesso
```

CMPSTA abilita le applicazioni perchè comparino i dati con qualsiasi altro banco. Dopo aver predisposto CMPVEC ($2C8), chiama GETCFG per convertire il numero del banco e JMP al codice nella RAM comune (condivisa) in $2BE che commuta i banchi, compara i dati, ristabilisce il vostro banco e ritorna al programma principale. Quando chiama un banco non di sistema, l'utente dovrebbe prendere le precauzioni necessarie per assicurarsi che le interruzioni (IRQ e NMI) saranno trattate in modo adeguato (o disabilitate prima).

19. $FF7D PRIMM ;stampa le utility immediate

PREPARAZIONE:

Registri:	nessuno
Memoria:	nessuna
Flag:	nessuno
Chiamate:	nessuna

RISULTATI:

Registri:	nessuno
Memoria:	nessuna
Flag:	nessuno

ESEMPIO:

```
JSR $FF7D       ;visualizza il testo successivo
.BYTE "messaggio
.BYTE $00       ;terminatore

JMP continua    ;l'esecuzione riprende qui
```

PRIMM è un'utility Kernal utilizzata per stampare (nell'unità di output per default) una stringa ASCII che segue immediatamente la chiamata. La stringa non deve essere più lunga di 255 caratteri ed è conclusa da un carattere zero ($00). Non può contenere caratteri zero incorporati. Poichè PRIMM usa lo stack del sistema per trovare la stringa ed un indirizzo di ritorno, non dovete JMP a PRIMM. Deve esserci un indirizzo valido sullo stack.

NUMERI DELLE UNITÀ DEL C128

Ecco i numeri delle unità del Commodore 128:

0 → Tastiera
1 → Registratore
2 → RS-232

3 → Schermo (corrente)
4 → Unità del bus seriale:
    4-7 di solito stampanti
    8-30 di solito dischetti

Il numero di dispositivo 31 non dovrebbe essere usato. Mentre viene specificato che esso è un indirizzo valido del bus seriale, quando subisce l'OR con certi comandi seriali, risulta un comando scorretto e sospende il bus ed i driver seriali.

GESTIONE DELLA MEMORIA NEL COMMODORE 128

MODO COMMODORE 128

In modo Commodore 128, tutta l'organizzazione della gestione della memoria dipende dalla configurazione della memoria attualmente selezionata. In BASIC C128 e nel Monitor in Linguaggio Macchina, la memoria è organizzata in 16 configurazioni di default della memoria. Sono presenti diverse porzioni di memoria a seconda della configurazione della memoria. La Figura 14-3 lista le configurazioni per default della memoria del C128 all'accensione del sistema.

BANCO	CONFIGURAZIONE
0	solo RAM(0)
1	solo RAM(1)
2	solo RAM(2) (la stessa di 0)
3	solo RAM(3) (la stessa di 1)
4	ROM Interna, RAM(0), I/O
5	ROM Interna, RAM(1), I/O
6	ROM Interna, RAM(2), I/O (la stessa di 4)
7	ROM Interna, RAM(3), I/O (la stessa di 5)
8	ROM Esterna, RAM(0), I/O
9	ROM Esterna, RAM(1), I/O
10	ROM Esterna, RAM(2), I/O (la stessa di 8)
11	ROM Esterna, RAM(3), I/O (la stessa di 9)
12	ROM Kernal e Interna (BASSA), RAM(0), I/O
13	ROM Kernal ed Esterna (BASSA), RAM(0), I/O
14	ROM BASIC e Kernal, RAM(0), Carattere ROM
15	ROM BASIC e Kernal, RAM(0), I/O

Figura 14-3. Tabella della Configurazione di Memoria (Banco)

L'UNITÀ DI GESTIONE MMU DELLA MEMORIA

In modo Commodore 128, tutta la gestione della memoria è controllata attraverso una serie di registri di I/O chiamati Unità di Gestione della Memoria (**Memory Management Unit, MMU**). L'Unità di Gestione della Memoria consiste in locazioni di memoria che si trovano tra $D500 e $D50B e $FF00 e $FF04. Il registro della configurazione appare due volte, una in $D500 ed ancora in $FF00. Questo viene fatto nel caso in cui l'I/O sia commutato fuori dall'intervallo di memoria $D000-$DFFF, che rende i registri dell'MMU inaccessibili da $D500-$D50B.

Quando questo accade, alcune funzioni dell'MMU non sono più disponibili per il programmatore. Così prima di disinserire l'I/O nell'intervallo $D000-$DFFF, assicuratevi di avere fatto tutte le operazioni di cui avete bisogno nell'MMU (in particolare, la preselezione dei valori dei registri delle preconfigurazioni per i registri di caricamento delle configurazioni). Consultate la Figura 13-4 per una rappresentazione grafica del modo in cui i registri MMU tracciano la mappa nella memoria.

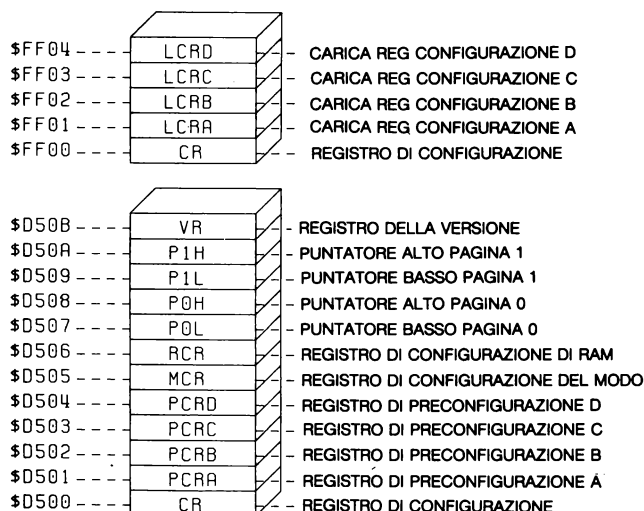



Figura 14-4. Mappa del Registro MMU

COME COMMUTARE I BANCHI

In BASIC, digitate il comando **BANK** per passare da un banco ad un altro come segue:

BANK n

dove n è un numero tra 0 e 15. Quando il vostro programma di applicazione ha bisogno di accedere ad un banco nel quale il microprocessore non è in contesto (per i comandi SYS, PEEK, POKE e WAIT) usate il comando BANK per raggiungere gli strati della memoria del computer che non sono accessibili nel banco corrente. Per esempio, supponete di volere accedere al chip VIC mentre state facendo dei grafici nella prima parte del vostro programma. Il segmento successivo del vostro programma ha bisogno di accedere alla ROM, che si può vedere solo nel banco 14 dell'8502. Cambiate nel banco 14; poi leggete i dati della ROM, che formano le immagini dei caratteri negli insiemi dei caratteri. Mentre il microprocessore sta "guardando" nel banco 14, il chip VIC non è disponibile per il microprocessore, così dovete emettere un altro comando BANK per ritornare a (una configurazione contenente l'I/O e) all'elaborazione dell'informazione del video VIC.

In linguaggio macchina, il commutare i banchi è un pò più difficile. Dovete modificare il valore dei registri [in particolare il **Registro di Configurazione** (**CR**), il **Registro Configurazione di Caricamento** (**LCR**) ed il **Registro di Preconfigurazione** (**PCR**)], sia direttamente che indirettamente. La routine Kernal GETCFG vi permette di cambiare le configurazioni e mantiene le stesse usate dal BASIC e dal Monitor come appaiono nella Figura 14-3. Ci sono quattro PCR e quattro LCR (come mostrato nella Figura 14-4). Ogni PCR corrisponde direttamente ad un LCR. PCR A corrisponde a LCR A, PCR B corrisponde a LCR

B e così via.

Per cambiare il valore del Registro di Configurazione direttamente, eseguite un'operazione di scrittura (STA, STX, STY) nel Registro di Configurazione. Per cambiare il valore del Registro di Configurazione indirettamente, scrivete un valore specificando una configurazione della memoria nel Registro di Preconfigurazione. Quando viene eseguita un'istruzione successiva di memorizzazione nel corrispondente Registro di Configurazione di Caricamento, il valore precedentemente memorizzato nel PCR corrispondente è caricato nel CR. Quando viene eseguita un'istruzione di memorizzazione in un LCR, il valore nel PCR corrispondente viene caricato nel CR e l'organizzazione della gestione della memoria si adatta ai valori associati a quello schema della gestione della memoria. Il valore scritto in LCR non ha conseguenze; qualsiasi valore genera il meccanismo della preconfigurazione.

NOTA: Il Basic aspetta che LCR e PCR siano lasciati soli. Se li usate per gestire la memoria nella vostra applicazione, non decidete di usare il BASIC.

IL REGISTRO DI CONFIGURAZIONE

Il Registro di Configurazione (CR) è il registro più importante dell'MMU. Esso specifica ed organizza la ROM, la RAM e le configurazioni di input/output per l'intera memoria del Commodore 128 in modo C128. (L'MMU non è presente nella mappa della memoria del C64). Il CR è localizzato all'indirizzo \$D500 quando l'I/O è disponibile ed all'indirizzo \$FF00 sempre. Quando le funzioni di I/O sono disabilitate, la memoria dell'MMU scompare tra \$D500 e \$D50B. La memoria dell'MMU è sempre presente tra \$FF00 e \$FF04. Ognuno degli otto bit in CR controlla una diversa funzione della memoria.

Il bit 0 (zero) nel Registro di Configurazione specifica se i Registri di I/O sono disponibili, o se la ROM (alta) è presente nell'intervallo della memoria \$D000 fino a \$DFFF. I Registri di I/O consistono in registri del chip VIC, chip SID, MMU (da \$D500 a \$D50B); numero 1 del CIA, che controlla la porta del joystick; e il numero 2 del CIA, che controlla il bus seriale e l'ingresso dell'utente. Se il bit 0 è alto (uguale a 1), la ROM o la RAM è presente nell'intervallo da \$D000 a \$DFFF, dipendendo dai valori dei bit della ROM ALTA (4 e 5) in questo registro. Se il bit 0 è basso (uguale a 0), l'I/O è presente in questo intervallo. Il valore del bit 0 all'accensione è 0.

Quando l'I/O è tolto (non è presente) in questo intervallo, i registri dell'MMU scompaiono dalla mappa della memoria di intervallo da \$D500 a \$D50B. Allora la gestione della memoria viene controllata dai registri MMU alle locazioni da \$FF00 a \$FF04. Qualsiasi operazione di scrittura in un LCR (\$FF01 – \$FF04) carica il valore PCR corrispondente (correntemente invisibile all'8502) nel CR. La lettura di un LCR riporta il valore memorizzato nel PCR correntemente inaccessibile. I registri dell'MMU che vanno da \$FF00 a \$FF04 sono sempre presenti nella memoria del Commodore 128, senza tenere conto della configurazione della memoria.

Il bit 1 in CR specifica come il microprocessore acceda all'intervallo di indirizzo da $4000 a $7FFF, chiamato memoria ROM BASSA (**ROM LOW**). Se il bit 1 è alto, il microprocessore accede alla RAM in questo intervallo. Se il bit 1 è basso (uguale a 0), il microprocessore traccia una mappa della **BASIC LOW ROM** in quell'intervallo. All'accensione o al reset, questo bit viene posto basso, così il BASIC è disponibile per l'utente non appena il computer viene acceso.

I bit 2 e 3 determinano il tipo di memoria che risiede nell'intervallo di mezzo della memoria, l'intervallo di indirizzo da $8000 a $BFFF. Se entrambi i bit 2 e 3 sono posti alti, la RAM viene posta in questo intervallo. Se il bit 2 è alto e il bit 3 è basso, viene posta qui la **INTERNAL FUNCTION ROM.** Se il bit 2 è basso ed il bit 3 è alto, appare la **EXTERNAL FUNCTION ROM.** Se i bit 2 e 3 sono bassi, viene posta qui la **BASIC HIGH ROM.** All'accensione o al reset, MMU pone entrambi i bit 2 e 3 bassi, così il BASIC è disponibile immediatamente per l'utente.

I bit 4 e 5 operano in modo simile ai bit 2 e 3 e specificano la memoria nell'intervallo da $C000 a $FFFF, a cui ci si riferisce come memoria **HIGH** (ALTA). Se i bit 4 e 5 sono posti alti, la RAM viene posta in questo intervallo. Se il bit 4 è alto ed il bit 5 è basso, viene posta la INTERNAL FUNCTION ROM. Se il bit 4 è basso ed il bit 5 è alto, appare la EXTERNAL FUNCTION ROM. Se i bit 4 e 5 sono bassi, vengono poste qui il Kernal e le ROM del carattere. Al momento dell'accensione o del reset, i bit 4 e 5 vengono posti bassi, così il Kernal e la ROM del carattere sono disponibili subito per l'utente.

Notate che il bit 0 del Registro di Configurazione, il bit che accende o spegne l'I/O dell'indirizzo di intervallo da $D000 a $DFFF, non tiene conto dell'organizzazione della memoria per i bit 4 e 5. Se il bit 0 è posto alto (1), i Registri di I/O non sono nell'indirizzo di intervallo $D000 fino a $DFFF. Sia la ROM del carattere, funzione interna od esterna della ROM, che la RAM sono localizzate in questo intervallo, dipendendo dal valore dei bit 4 e 5 del Registro di Configurazione. Se il bit 0 del Registro di Configurazione è posto basso (0), i registri di Input/Output sono presenti tra $D000 e $DFFF, senza tenere conto del valore dei bit 4 e 5 del registro di configurazione. Questo significa che non ha importanza quale configurazione di memoria venga scelta tra $C000 e $FFFF con i bit 4 e 5, se il bit 0 è posto basso (0), la ROM del carattere (o qualsiasi cosa ci fosse in origine in questo intervallo dell'indirizzo) viene coperto dai registri di I/O e non è più disponibile per il microprocessore. Ecco perchè la ROM del carattere ed i registri di I/O non sono mai disponibili nello stesso momento.

Infine, gli ultimi due bit dell'MMU, 6 e 7, determinano la selezione del **RAM BANK** (banco della RAM). Per il sistema base di 128K, solo il bit 6 è significativo; il bit 7 non è implementato. Quando il bit 6 è alto (1), viene selezionato il banco 1 della RAM. Quando il bit 6 è basso (0), è selezionato il banco 0 della RAM.

I 128K della RAM sono organizzati in due banchi da 64K della RAM. Il microprocessore indirizza solo 64K per volta, ma poichè i due banchi da 64K della RAM possono essere inseriti e disinseriti velocemente, il computer si comporta come se indirizzasse 128K nello stesso momento. Quando il microprocessore si indirizza ad un banco della RAM, l'altro banco memorizza l'informazione da elaborare una volta che viene inserita nel banco. Parti di entrambi i banchi possono essere divise nella memoria contemporaneamente. Questa è chiamata RAM Comune ed è trattata nella sezione Il Registro di Configurazione della RAM. Il banco 0 della RAM è usato di solito per un'area di testo in BASIC, mentre il banco 1 della RAM è usato per matrici in BASIC e per la memorizzazione di variabili.

Come indicato sopra, l'MMU ha una funzione che permette ad una porzione della RAM di essere in comune ai due banchi della RAM. Il Registro di Configurazione RAM controlla la quantità di RAM comune. Ciò verrà trattato in dettaglio più tardi in questa sezione nella descrizione del Registro di Configurazione RAM.

La Figura 14-5 è un diagramma del Registro di Configurazione, che mostra come i bit controllino ogni organizzazione della memoria.

7 6	5 4	3 2	1	0
SELEZIONE DEL BANCO	SPAZIO ALTO ($C000-$FFFF)	SPAZIO MEDIO ($8000-$BFFF)	SPAZIO BASSO ($4000-$7FFF)	SPAZIO DI I/O ($D000-$DFFF)
00 = Banco RAM 0 01 = Banco RAM 1 10 = Banco espans. 2 11 = Banco espans. 3	00 = ROM Kernal 01 = ROM funz. interna 01 = ROM funz. esterna 11 = RAM	00 = ROM alta Basic 01 = ROM funz. interna 10 = ROM funz. esterna 11 = RAM	0 = ROM bassa Basic 1 = RAM	0 = Registri di I/O I = RAM/ROM

Figura 14-5. Registro di Configurazione

PRECONFIGURAZIONE DELLA MEMORIA DEL COMMODORE 128

Come notato, il Registro di Configurazione dell'MMU è il mezzo con il quale il Commodore 128 organizza il tracciato della memoria. La gestione della memoria del Commodore 128 ha un meccanismo che permette al programmatore di predisporre quattro configurazioni di memoria predefinite oltre a quella già operante. I quattro Registri di Preconfigurazione (PCR) ed i quattro Registri della Configurazione di Caricamento (LCR) sono programmati per fornire questa funzione. Se si desidera una diversa struttura di gestione della memoria, si può scegliere facilmente ed all'istante una configurazione precedentemente definita in uno dei Registri di Preconfigurazione.

Questo meccanismo permette al programmatore di prestabilire parecchie configurazioni di memoria e, con una sola istruzione di memorizzazione ad un LCR, di riorganizzare l'intero tracciato della memoria all'istante. Questo dà la possibilità all'utente di usare parecchie organizzazioni di memoria in modo intercambiabile; così, se una parte del vostro programma richiede una configurazione di memoria ed un'altra parte richiede una diversa configurazione, potete scambiare le due, ognuna con una singola istruzione di memorizzazione, una volta che predefinite la predisposizione alternata in memoria in uno dei Registri di Preconfigurazione.

Per alterare la gestione della memoria con questo meccanismo della preconfigurazione, dovete cambiare il valore del Registro di Preconfigurazione. I quattro PCR ed i quattro LCR dell'MMU sono mostrati nella Mappa del Registro MMU

nella Figura 14-6. Ogni PCR corrisponde direttamente ad un LCR (cioè, PCR A corrisponde a LCR A, PCR B corrisponde a LCR B e così via). Il formato per il Registro di Configurazione di Caricamento e per il Registro di Preconfigurazione è lo stesso che per il Registro di Configurazione. I quattro LCR ed i quattro PCR sono tutti inizializzati a zero.

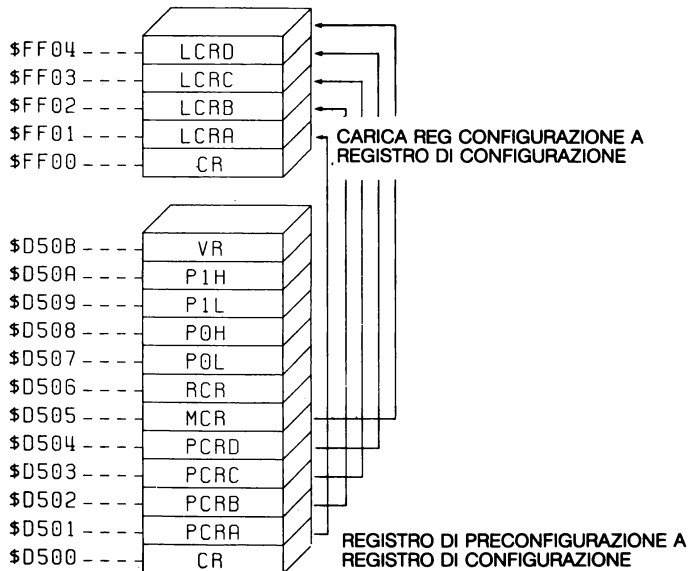



Figura 14-6. Mappa del Registro MMU

Per cambiare direttamente il valore del Registro di Configurazione (passando perciò oltre il meccanismo di preconfigurazione), eseguite un'operazione di scrittura (STA, STX, STY) direttamente nel Registro di Configurazione o in $D500 o in $FF00.

Per cambiare indirettamente il valore del Registro di Configurazione (utilizzando perciò il meccanismo di preconfigurazione), eseguite un'operazione di scrittura (STA, STX, STY) nel Registro di Preconfigurazione. Esso carica un valore nel PCR. Quando viene eseguita un'operazione successiva di memorizzazione nel Registro di Configurazione di Caricamento corrispondente, il valore che precedentemente era stato memorizzato nel PCR corrispondente è caricato in CR. L'istruzione di caricamento agisce come meccanismo a grilletto che passa i contenuti di PCR in CR. Quando viene eseguita un'istruzione di memorizzazione in un LCR, il valore nel PCR corrispondente viene caricato nel CR e l'organizzazione della gestione della memoria è conforme al valore associato a quel PCR.

Per esempio, per alterare l'organizzazione della gestione della memoria specificata al momento dell'accensione, usate il seguente segmento in linguaggio macchina:

```
PART 1 LDA #$0E        Carica l'accumulatore con 14
                       Questo seleziona I/O ($D000—$DFFF)
                       RAM ($4000—$7FFF, $8000—$BFFF)
                       Kernal e ROM ($C000—$FFFF)
                       RAM banco 0
       STA $D501       Carica in PCR A— senza risultati immediati:
       :
       :               Pone qui la parte interim del programma
PART 2 STA $FF01       Scrive in LCR A, seleziona la configurazione di
                       cui sopra
```

In questo segmento di programma, PART 1 inizializza PCR A ($D501) con il valore 14 ($0E), che non dà risultati immediati. Quando si incontra la PART 2, l'istruzione STA esegue un'operazione di scrittura in locazione $FF01, che lancia il meccanismo della preconfigurazione e carica il valore da PCR A nel Registro di Configurazione. Il valore dell'istruzione di caricamento, non è significativo; deve operare solo nell'indirizzo e qualsiasi istruzione di memorizzazione funzionerà. Una volta che l'istruzione viene eseguita, l'organizzazione di gestione della memoria viene modificata subito secondo il valore del PCR appropriato, in questo caso PCR A ($D501).

Il valore 14 ($0E) caricato in PCR A seleziona la seguente organizzazione di memoria:

TIPO	**INTERVALLO DI INDIRIZZO**	**BIT COINVOLTI IN CR**
I/O	**$D000-$DFFF**	**0**
RAM	**$4000-$7FFF**	**1**
RAM	**$8000-$BFFF**	**2,3**
Kernal, Car	**$C000-$FFFF**	**4,5**
RAM BANCO 0	**—**	**6,7**
	14=$0E=0000 1110 (binario)	

È buona pratica inizializzare il PCR all'inizio del vostro programma, come in PART 1 del segmento di programma di cui sopra. Poi ponete in memoria la parte interim del vostro programma. La PART 2 segna il luogo del vostro programma dove state andando realmente per alterare la predisposizione della gestione della memoria. Per ottenere il massimo della velocità e dell'efficienza del processore, usate l'indirizzo assoluto per i registri MMU, come nell'esempio sopra.

Come notato precedentemente, quando i Registri di Input/Output sono spenti, i registri dell'MMU che appaiono nell'intervallo di memoria $D500-$D50B non sono disponibili per il microprocessore. Quando questo accade, un'operazione di scrittura in un LCR carica ancora il valore del PCR corrispondente (correntemente invisibile all'8502) in CR, anche se i PCR non sono accessibili al microprocessore quando gli I/O sono spenti. Se spegnete gli I/O, assicuratevi di porre prima i PCR, che sono presenti quando il bit 0 del Registro di Configurazione è uguale a 0. Tuttavia, i Registri della Configurazione di Caricamento

sono disponibili sempre poichè appaiono nell'intervallo di indirrizzo $FF00–$FF04. Se I/O è spento, PCR deve essere posto prima di spegnere I/O perchè siano utili. Notate che il BASIC usa i PCR; se li modificate quando c'è il BASIC, potete ottenere risultati imprevedibili.

I registri MMU controllano l'organizzazione della memoria per la RAM, il BASIC, il Kernal ed i caratteri ROM, le funzioni interne ed esterne della ROM e l'I/O. MMU ha ulteriori registri che determinano il modo (C128, C64 o CP/M) nel quale operano il Commodore 128, le configurazioni della RAM comune e la locazione delle pagine 0 e 1. I registri dell'MMU che controllano queste operazioni sono rispettivamente il **Registro di Configurazione del Modo (MCR)**, il **Registro di Configurazione della RAM** (**RCR**) ed i **Puntatori di Pagina.** La sezione seguente spiega come operino questi ulteriori registri dell'Unità di Gestione della Memoria.

IL REGISTRO DI CONFIGURAZIONE DEL MODO

Il Registro di Configurazione del Modo (MCR) specifica quale microprocessore operi correntemente (8502, Z80A) e quale modo del sistema operativo venga chiamato (C128 o C64). L'MCR è localizzato all'indirizzo $D505. Come negli altri registri nell'MMU ogni bit del Registro di Configurazione del Modo controlla un'operazione diversa ed indipendente.

Il bit 0 determina quale microprocessore abbia il controllo del Commodore 128. Il bit 0 viene inserito basso così il microprocessore Z80 inizializza il controllo del computer. Lo Z80 esegue una piccola procedura di avviamento iniziale, poi il bit 0 è posto a 1 e l'8502 assume il controllo se non è presente il dischetto del sistema CP/M nel drive.

Se il Commodore 128 viene inserito per primo o risettato ed il disk drive riconosce il dischetto del sistema operativo CP/M nel drive, il microprocessore Z80A fa il BOOT del sistema operativo CP/M dal dischetto. Il valore del bit 0 nel Registro di Configurazione del Modo è in questo caso 0. Quando lo Z80A prende il controllo del Commodore 128, tutti i riferimenti di memoria da $0000 a $0FFF sono spostati da $D000 a $DFFF, dove esiste il CP/M BIOS nella ROM. Per accessi alla memoria nell'intervallo da $0000 a $0FFF nello Z80 BIOS, le linee di stato della memoria MS0 e MS1 sono poste basse per riflettere la ROM; altrimenti sono alte. Notate che il modo C64 ed il modo Z80A sono una configurazione indefinita.

I bit 1 e 2 non vengono usati. Essi sono riservati per espansioni future. IF0, bit 3 pone un input per FAST seriale. Non è assolutamente usato come porta di input.

Il bit 3 è il bit di controllo del disk drive seriale fast (veloce) (FSDIR). Si comporta come un bit di una porta bidirezionale 6529, cioè si comporta in modo diverso a seconda se il bit viene usato per un'operazione di input o di output. Come segnale di output, il bit 3 controlla la direzione dei dati nel buffer dati del disk drive. Il pin FSDIR dell'MMU riflette lo stato del bit 3, che viene rimesso a zero al momento dell'accensione. Se il bit 3 è uguale a 1, avviene un'operazione di output, che seleziona una direzione dei dati per i dati nel buffer del bus seriale. Se è zero, il bit 3 pone un input per il FAST seriale. Non viene assolutamente usato come porta di input.

I bit 4 e 5 sono rispettivamente i bit di lettura /GAME ed /EXROM. Queste linee di controllo della cartuccia inizializzano il modo Commodore 64 ed operano come linee hardware /GAME ed /EXROM come nel Commodore 64. Quando queste linee di controllo riconoscono una cartuccia nella porta di espansione del Commodore 128, viene abilitato all'istante il modo C64, il computer agisce come un Commodore 64 e prende le istruzioni dal software della cartuccia. Al momento dell'inserimento, /GAME ed /EXROM vengono prelevati come input. Se uno è basso, viene selezionato il modo C64. Così, una cartuccia C128 non dovrebbe prelevare queste linee come basse al momento dell'inserimento. In modo C128, queste linee sono attive come linee di I/O (trattenute) nella porta di espansione. Esse possono essere utilizzate come linee di input o output, ma assicuratevi che non siano basse al momento dell'inserimento in modo C128.

Il bit 6 seleziona il sistema operativo che assume il controllo del Commodore 128. Al momento dell'accensione o del reset, questo bit è cancellato (0) per abilitare tutti i registri dell'MMU e le funzioni del modo Commodore 128. Ponendo questo bit alto (1) si inizializza il modo C64.

Il bit 7, un bit a solo lettura, indica se il tasto **VIDEO 40/80** è nella posizione in su (40 colonne) o in giù (80 colonne). Il valore del bit 7 è alto (1) se il tasto **40/80** è nella posizione in su. Il valore del bit 7 è basso (0) se il tasto **VIDEO 40/80** è nella posizione in giù.

Questo è utile in certi programmi di applicazione che utilizzano sia il video a 40 colonne che quello ad 80 colonne. Per esempio, verificate il valore di questo bit per vedere se l'utente visualizza lo schermo VIC. Se è così, proseguite con il programma; altrimenti visualizzate un messaggio che dice all'utente di passare dallo schermo ad 80 colonne allo schermo VIC (40 colonne) per visualizzare un grafico VIC. Ciò può non essere valido se l'utente ha digitato <ESC>X per commutare gli schermi. Consultate la routine Kernal SCORG.

Consultate la Figura 14-7 per un riassunto delle attività del Registro di Configurazione del Modo.

$D505 →

7	6	5	4	3	2	1	0

Registro di Configurazione del Modo

BIT	DESCRIZIÓNE DELLA FUNZIONE	VALORE ALTO	VALORE BASSO
0	/Seleziona microprocessore	8502	Z80A (invertito)
1	Non usato	— —	— —
2	Non usato	— —	— —
3	Controllo seriale veloce DD	seriale veloce out	seriale veloce in
4	/GAME cartuccia di accesso gioco (solo modo C64)		modo C64 all'accens
5	/EXROM cartuccia software di accesso esterno (solo modo C64)		modo C64 all'accens
6	Selezione sistema operativo	modo C64 (MMU scompare da mappa memoria)	modo C128 pone i registri MMU
7	Legge posizione tasto 40/80 e	40/80 colonne tasto SU	40/80 colonne tasto GIU

Figura 14-7. Riassunto del Registro di Configurazione del Modo

IL REGISTRO DI CONFIGURAZIONE DELLA RAM

Il Registro di Configurazione della RAM (RCR) nell'MMU specifica la quantità di RAM Comune condivisa tra i due banchi 64K della RAM, come la RAM è divisa e quale banco è assegnato per il chip VIC. Il valore dei bit nel RCR determina il modo in cui ogni banco della RAM è assegnato per scopi specifici. Il Registro di Configurazione della RAM è localizzato all'interno del blocco di I/O all'indirizzo $D506.

I bit 0 e 1 determinano la quantità di RAM condivisa tra i banchi. Se entrambi i bit 0 e 1 sono uguali a 0, 1K della RAM diventa in comune. Se il bit 0 è uguale a 1 (alto) ed il bit 1 è basso (0), allora 4K della RAM comune vengono condivisi tra i banchi. Se il bit 0 è basso (0) ed il bit 1 è alto (1), allora 8K sono in comune e se entrambi i bit 2 e 3 sono alti (1), allora 16K della RAM sono in comune. (Questi bit non hanno effetto in modo Commodore 64). I valori di reset di questi bit sono entrambi 0. Consultate la Figura 14-8 per capire come i due banchi della RAM condividono la RAM.

I bit 2 e 3 specificano quali porzioni dei due banchi della RAM sono in comune, se ci sono. Se entrambi i bit sono bassi (0), non c'è condivisione della RAM. Se il bit 2 è posto alto (1) ed il bit 3 è basso (0), una parte della parte bassa del banco 0 della RAM sostituisce la parte corrispondente del banco 1 della RAM per tutti gli accessi agli indirizzi della RAM. Se il bit 3 è posto alto (1) ed il bit 2 è posto basso (0), una parte del banco 0 della parte alta della RAM sostituisce la parte corrispondente del banco 1 della RAM per tutti gli accessi agli indirizzi della RAM. Se entrambi i bit 2 e 3 sono 1 (alti), la RAM è in comune ad entrambe le parti alta e bassa dei banchi della RAM. All'accensione o al reset, i bit 2 e 3 sono posti a 0, e non viene condivisa la RAM tra i banchi. Da un punto di vista hardware, i 128K MMU selezionano la RAM comune forzando la linea di abilitazione CAS0 bassa e la linea di abilitazione CAS1 alta per tutti gli accessi della memoria comune.

Ai bit 4 e 5 non è stata assegnata alcuna funzione. Essi sono riservati per espansioni future.

I bit 6 e 7 nell'RCR operano come un puntatore del banco della RAM che dice al chip VIC quale porzione della RAM usare. Al momento il bit 7 viene ignorato. Anch'esso è riservato per una futura espansione della RAM. Quando il bit 6 è basso (0) (pilotando CAS0 basso) viene detto al chip VIC di guardare nel banco 0 della RAM. Quando il bit 6 è posto alto (1) (pilotando CAS1 basso), il chip VIC viene diretto nel banco 1 della RAM. Entrambe le predisposizioni permettono al banco della RAM del chip VIC di essere selezionato dal banco RAM del microprocessore in modo indipendente.

Quando la velocità del microprocessore viene aumentata a 2MHz, il chip VIC è disabilitato ed il chip ad 80 colonne (8563) assume il controllo dell'elaborazione a video. Il chip VIC viene coinvolto tenendo la linea AEC hardware alta. La disabilitazione del chip VIC non è coinvolta direttamente dalle azioni dell'MMU. La Figura 14-9 riassume le attività del Registro di Configurazione della RAM.

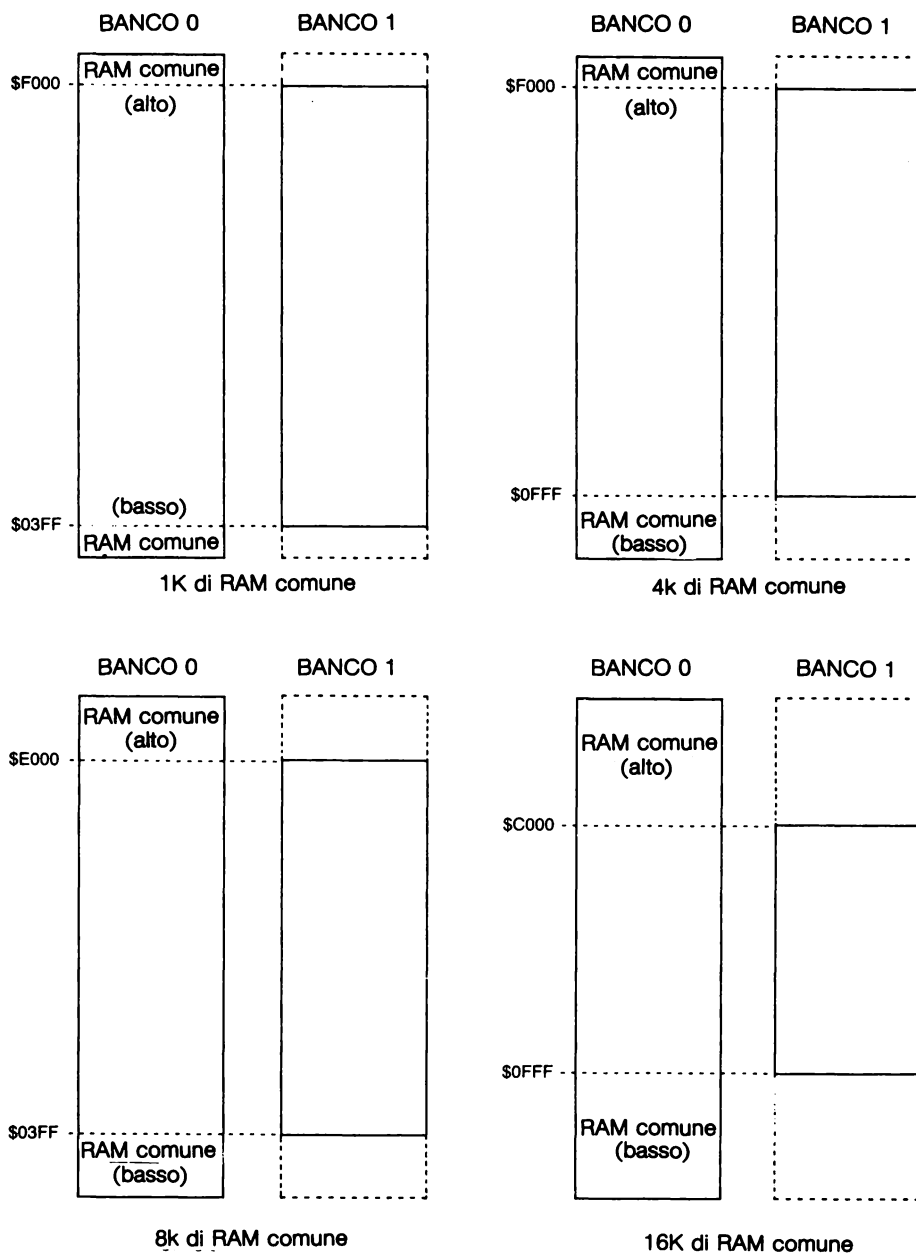



Figura 14-8. RAM comune

$D506 → | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Registro di Configurazione della RAM

BIT	DESCRIZIONE FUNZIONALE
1 0	**– Determinano la quantità di RAM condivisa tra i banchi della RAM**
0 0	**= 1K di RAM comune**
0 1	**= 4K di RAM comune**
1 0	**= 8K di RAM comune**
1 1	**= 16K di RAM comune**
3 2	**– Determinano quale porzione di RAM viene divisa**
0 0	**= No RAM comune**
0 1	**= Il basso del banco 0 RAM è comune**
1 0	**= L'alto del banco 0 RAM è comune**
1 1	**= Alto e basso del banco 0 RAM sono comuni**
5 4	**– Non implementati; riservati a future espansioni**
7 6	**– Seleziona il banco RAM per il chip VIC (il bit 7 è ignorato)**
X 0	**= Seleziona il Banco 0 per il chip VIC**
X 1	**= Seleziona il Banco 1 per il chip VIC**

Nota: X = non implementato

Figura 14-9. Riassunto del Registro della Configurazione della RAM

I PUNTATORI DI PAGINA

Il Commodore 128 ha una funzione che vi permette di rilocalizzare la pagina 0 ($00–$FF) e la pagina 1 ($100–$1FF) della memoria. Certe applicazioni possono richiedere che voi teniate la pagina 0 intatta mentre lanciate il BASIC, spegnete il BASIC ROM, riprendete il procedimento all'interno del controllo del vostro programma in linguaggio macchina e poi accendete nuovamente la pagina 0 e BASIC ROM. Invece di trasferire la pagina 0 con il monitor in linguaggio macchina, i puntatori delle pagine rendono facile rilocalizzare le pagine 0 e 1. Quattro registri all'interno dell'MMU, chiamati **puntatori di pagina**, vi permettono di fare ciò. I puntatori di pagina sono localizzati all'interno del blocco di I/O nell'intervallo da $D507 a $D50A. I puntatori di pagina seguono il formato standard 8502 byte basso/byte alto. Ecco gli indirizzi reali dei byte alto e basso dei puntatori di pagina:

$D507 Puntatore della Pagina 0 (basso)
$D508 Puntatore della Pagina 0 (alto)
$D509 Puntatore della Pagina 1 (basso)
$D50A Puntatore della Pagina 1 (alto)

Il bit 0 dei puntatori di pagina a byte alto corrisponde al numero di banco della RAM per qualsiasi accesso all'indirizzo della pagina 0. Il bit 0 controlla la generazione della linea di controllo CAS0 hardware se è basso e della linea CAS1 se è alto ed i bit 1 e 3 vengono ignorati.

Per rilocare la pagina 0, eseguite un'operazione di scrittura sul puntatore a byte alto della pagina 0. Questo è memorizzato nella locazione del puntatore di pagina a byte alto e non ha risultati diretti finchè non viene eseguita un'operazione di scrittura nel puntatore della pagina 0 a byte basso. Quando questo avviene, i bit da 0 a 7 del puntatore della pagina a byte basso corrispondono alle linee di Indirizzo Tradotto da TA8 a TA15 per ogni riferimento all'indirizzo della pagina 0, che riloca la pagina 0. Qualsiasi indirizzo successivo della pagina 0 viene ritrasmesso alla nuova pagina zero.

Il puntatore della pagina 1 funziona nello stesso modo. Entrambe le coppie di puntatori vengono inizializzati a 0 al momento dell'accensione, ponendo le pagine 0 e 1 nelle locazioni effettive di pagina 0 e 1.

È importante notare che gli indirizzi 0 e 1 della memoria sono sempre disponibili per quei riferimenti assoluti dell'indirizzo, senza tenere conto se le pagine zero e uno sono rilocate.

Il puntatore della pagina zero a byte basso ripone direttamente l'indirizzo di ordine più elevato della pagina zero (normalmente 00). Quando le pagine zero ed uno sono dirette alle locazioni nella memoria della RAM sopra la pagina uno, MMU traduce gli indirizzi di nuovo nelle normali locazioni delle pagine zero ed uno, scambiando effettivamente quelle due pagine della memoria. Questa traduzione dell'indirizzo si applica solo alla RAM; i registri della ROM e di I/O non possono essere ritradotti. Gli indirizzi del chip VIC non sono ritradotti nelle loro locazioni originarie della memoria, così dovete assicurarvi di non porre la pagina zero o uno nell'indirizzo del chip VIC.

La ROM che appare in questi intervalli di indirizzo copre ancora la RAM senza tenere conto se la RAM è a pagina zero, pagina uno o qualsiasi altra pagina della memoria della RAM. Se avete bisogno di usare il Kernal, le variabili necessarie richieste per la routine Kernal devono essere poste in memoria dove il Kernal è in contesto, per esempio il banco 15 ($0F) del Monitor di Linguaggio Macchina.

Il bit 0 di entrambi i puntatori di pagina a byte alto (0 e 1) corrisponde al numero di banco della RAM per qualsiasi accesso all'indirizzo della pagina zero o uno. La pagina zero a byte alto (bit 0) di solito non tiene conto del banco della RAM posto dal registro di configurazione. Tuttavia, se la RAM comune (in fondo al banco 0 della RAM) viene specificata, il puntatore a byte alto per le pagine zero ed uno viene ignorato e le pagine zero e uno appaiono nella RAM comune. Altrimenti, se la RAM comune non è assegnata, le pagine zero ed uno appaiono dove voi le specificate, secondo i contenuti dei puntatori di pagina. In altre parole, la RAM comune ha la priorità sui puntatori di pagina, se la RAM comune è assegnata dal registro di configurazione della RAM.

Questa funzione della rilocazione della pagina zero e della pagina uno dà molti vantaggi al programmatore. Questo permette ai programmi in linguaggio macchina di creare parecchie pagine di variabili di pagina zero o parecchi stack diversi. Quando avete bisogno di accedere alle variabili ulteriori di pagina zero o ai 256 byte extra dello stack, modificate semplicemente il puntatore in modo che indichi la pagina seguente. Questo dà un'ulteriore velocità ai vostri programmi poichè potete usare l'indirizzamento alla pagina zero per pagine zero succes-

sive. Potete anche porre la pagina zero nella memoria dello schermo per scritture veloci extra nello schermo. Inoltre, vi dà un modo per implementare livelli più profondi delle subroutine poichè avete un'area di stack più grande. Ricordate, però, di lasciare tre byte in cima allo stack per richieste di interruzione e di servizio.

IL REGISTRO DELLA VERSIONE DEL SISTEMA

Il Registro della Versione del Sistema, localizzato all'indirizzo $D50B, contiene un valore che dice all'utente quale versione dell'MMU è all'interno del C128 e quanto è grande la memoria. I bit da 0 a 3 contengono il numero della versione MMU. I bit da 4 a 7 contengono un valore (nei blocchi della memoria) secondo la grandezza della memoria del C128. Questo permette al programmatore di controllare la versione del C128 e la grandezza della memoria e di renderla compatibile coi sistemi che saranno ampliati in futuro. La versione corrente del C128 contiene il valore $20, che significa due blocchi da 64K.

AUTO AVVIAMENTO DI UNA CARTUCCIA DI APPLICAZIONE ROM

Molti di voi possono voler porre il loro programma di applicazione in una cartuccia che si innesta nella porta di espansione. Per far partire automaticamente il programma non appena accendete il computer, dovete porre una particolare sequenza codificata nei primi 6 byte dove la cartuccia esterna (o interna) della ROM si innesta nella memoria. Ecco la sequenza di auto avviamento sia in modo C128 che in modo C64.

MODO COMMODORE 128

Qualsiasi cartuccia C64 è presente automaticamente se il sistema riconosce che /GAME o /EXROM sono posti bassi.
Se qualsiasi cartuccia C128 viene installata nella porta di espansione:

1. Registra I.D. della cartuccia nella Tabella di Indirizzo Fisico (PAT).
2. Se I.D. è uguale a 1, chiama il vettore a partenza a freddo (che può fare l'RTS).

I primi 9 byte della sequenza di auto avviamento del Commodore 128 sono:

BYTE

$X000 Vettore a partenza a freddo
$X003 Vettore a partenza a caldo (non usato ma deve essere specificato)
$X006 ID della Cartuccia I.D.(L'I.D. = 1 per una scheda auto iniziante)
$X007 Il Carattere ASCII "C" con il bit alto
$X008 Il Carattere ASCII "B" con il bit alto
$X009 Il Carattere ASCII "M" con il bit alto

dove X è la cifra esadecimale "8" per $8000 o "C" per $C000.

Ci sono quattro fessure dove le cartucce (ROM) possono inserirsi (due interne, due esterne). Esse devovo seguire la sequenza descritta sopra, sia che siano interne che esterne.

IL MODO COMMODORE 64

I primi 10 byte della sequenza di auto avviamento del Commodore 64 sono:

BYTE

$X000 Vettore a partenza a freddo
$X003 Vettore a partenza a caldo
$X006 Il Carattere ASCII "C" con il bit alto
$X007 Il Carattere ASCII "B" con il bit alto
$X008 Il Carattere ASCII "M" con il bit alto
$X009 Il Carattere ASCII "8" con il bit alto
$X00A Il Carattere ASCII "0" con il bit alto

dove X è la cifra esadecimale "8" per $8000 o "C" per $C000.

L'header della cartuccia che viene raccomandato per entrambi i modi operazionali è il seguente:

```
SEI
JMP START
NOP
NOP
.
.
.
```

Questo header viene raccomandato perché il bit di stato dell'interruzione della disabilitazione sia posto quando il controllo passa al software nella cartuccia della ROM.

L'EDITOR DELLO SCHERMO DEL COMMODORE 128

L'editor dello schermo del Commodore è tra i più facili da usare. Non appena accendete il computer, l'editor dello schermo è disponibile. Non dovete chiamare ulteriori editor di testo. Usando i tasti per manipolare il testo, l'editor dello schermo vi dà più libertà rispetto a molti altri editor.

CODICI DI ESCAPE NELL'EDITOR DEL C128

Per usare le seguenti funzioni **ESCAPE,** premete il tasto ESCAPE e poi premete il tasto della funzione che volete.

TASTO	FUNZIONE
A	Abilita il modo auto inserimento
B	Pone la parte in basso a destra della finestra dello schermo nella posizione del cursore
C	Disabilita il modo auto inserimento
D	Cancella la linea corrente
E	Pone il cursore in modo non lampeggiante
F	Pone il cursore in modo lampeggiante
G	Abilita il suono (control-G)
H	Disabilita il suono
I	Inserisce una linea
J	Si muove all'inizio della linea corrente
K	Si muove alla fine della linea corrente
L	Abilita lo scrolling
M	Disabilita lo scrolling
N	Riporta lo schermo allo stato normale (non invertito) (solo per 80 colonne)
O	Modi cancellamento, inserimento, delimitazione, sottolineatura, lampeggio ed inversione)
P	Cancella fino all'inizio della linea corrente
Q	Cancella fino alla fine della linea corrente
R	Pone lo schermo in inversione (solo schermo ad 80 colonne)
S	Cambia in cursore blocco (solo schermo ad 80 colonne)
T	Pone la parte in alto a sinistra della finestra dello schermo nella posizione del cursore
U	Cambia in cursore sottolineatore (solo schermo ad 80 colonne)
V	Scrolling in su
W	Scrolling in giù
X	Scambia l'unità di output video 40/80 colonne
Y	Pone, per default, gli stop del tabulatore (8 spazi)
Z	Azzera tutti gli stop del tabulatore
@	Azzera fino alla fine dello schermo

CODICI DI CONTROLLO DELL'EDITOR DEL C128

I seguenti caratteri di controllo nella tabella CBM ASCII sono stati aggiunti o modificati rispetto a quelli trovati nel C64. I codici che non vengono mostrati in questa tabella hanno la stessa funzione di quella che avevano nel C64.

VALORE CHR$	CONTROLLO TASTIERA	FUNZIONE DEL CARATTERE
2	B	Sottolineatura ON (accesa) (solo schermo ad 80 colonne)
7	G	Produce i suoni
9	I	Carattere del tabulatore
10	J	Carattere di salto riga
11	K	Disabilita lo shift + C= (già codice 9)
12	L	Abilita lo shift + C= (già codice 8)
15	O	Accende il lampeggio (solo schermo ad 80 colonne)
24	X	Pone/azzera il tabulatore
27	[	Carattere di cambio codice
130		Sottolineatura OFF (spenta) (solo schermo ad 80 colonne)
143		Spegne il lampeggio (solo schermo ad 80 colonne)

TABELLA DI SALTO DELL'EDITOR DEL C128

Le chiamate dell'editor listate qui sotto sono un insieme di estensioni della tabella di salto standard CBM. Esse sono specifiche per il C128 e non dovrebbero essere considerate come aggiunte permanenti alla tabella di salto standard. Esse sono tutte vere subroutine e terminano con l'istruzione RTS. Come per tutte le chiamate Kernal, la configurazione del sistema (ROM alta, RAM-0 e I/O) devono essere in contesto al momento della chiamata.

1.	$C000 CINT	;inizializza l'editor e lo schermo
2.	$C003 DISPLY	;visualizza .A=car, .X=colore
3.	$C006 LP2	;ottiene un tasto dal buffer irq in .A
4.	$C009 LOOP5	;ottiene un chr dalla linea dello schermo in .A
5.	$C00C PRINT	;stampa un carattere in .A
6.	$C00F SCRORG	;ottiene la misura della finestra corrente
7.	$C012 SCNKEY	;analizza la subroutine della tastiera
8.	$C015 REPEAT	;ripete il tasto logico e CKIT2
9.	$C018 PLOT	;legge o pone la posizione del cursore
10.	$C018 CURSOR	;muove la subroutine del cursore dell'8563
11.	$C01E ESCAPE	;esegue la funzione escape
12.	$C021 KEYSET	;ridefinisce un tasto programmabile
13.	$C024 IRQ	;entrata irq
14.	$C027 INIT80	;inizializza l'insieme dei caratteri ad 80 colonne
15.	$C02A SWAPPER	;cambio modo 40/80
16.	$C02D WINDOW	;pone UL o BR della finestra
17.	$C033 LDTB2	;tabella delle linee dello schermo a byte basso
18.	$C04C LDTB1	;tabella delle linee dello schermo a byte alto
19.	$334 CONTRL	;stampa l'indiretto CTRL
20.	$336 SHIFTD	;stampa l'indiretto SHFT
21.	$338 ESCAPE	;stampa l'indiretto ESC
22.	$33A KEYVEC	;indiretto logico del keyscan
23.	$33C KEYCHK	;indiretto della memorizzazione del keyscan
24.	$33E DECODE	;vettori della tabella di decodificazione della tastiera

Le entrate 17, 18 e 24 sono puntatori della tabella e non sono routine richiamabili. Le entrate 19—23 sono considerate vettori indiretti, non veri punti di entrata.

Questo capitolo ha presentato il sistema operativo del Commodore 128. Il capitolo 17, Volume settimo fornisce informazioni del CP/M delle mappe di memoria del Commodore 128 e del Commodore 64.

15

CP/M 3.0 SUL COMMODORE 128

Il CP/M® è un sistema operativo per microprocessore prodotto da Digital Research, Inc. (DRI). La versione del CP/M usata sul Commodore 128 è CP/M Plus® Versione 3.0. In questo capitolo, ci si riferisce generalmente al CP/M come CP/M 3.0 o semplicemente CP/M. Questo capitolo riassume gli aspetti del CP/M non dipendenti dal C128 sul Commodore 128. Per informazioni dettagliate sul CP/M 3.0 dipendente dal C128, consultate l'Appendice K nell'ottavo volume di questa *Guida.*

FABBISOGNI PER UN SISTEMA CP/M 3.0

I fabbisogni generali per il CP/M 3.0 sono:

- Un computer contenente un microprocessore Z80.
- Una console consistente di una tastiera e di uno schermo.
- Almeno un floppy disk drive.
- Il software del sistema CP/M su dischetto.

Il CP/M 3.0 sul Personal Computer Commodore 128 di solito è formato dai seguenti elementi:

- Un microprocessore Z80 incorporato.
- Una console formata dalla tastiera completa del Commodore 128 ed un monitor ad 80 colonne.
- Il disk drive veloce Commodore 1571
- Il disco del sistema CP/M, che include il sistema CP/M 3.0, un programma di utility HELP esauriente ed un numero di altri programmi di utility.

> **NOTA:** Il CP/M può essere usato anche su un monitor a 40 colonne. Per vedere tutte le 80 colonne del video dovete fare lo scroll dello schermo in orizzontale premendo il tasto **CONTROL** ed il tasto **CURSORE** appropriato (a sinistra o a destra).

Il CP/M 3.0 può essere usato anche con il disk drive 1541. In questo caso, possono essere usati solo dischetti GCR a singola faccia e la velocità dell'operazione sarà approssimativamente un decimo della velocità raggiunta usando il disk drive 1571. (Consultate la trattazione dei formati dei dischetti in questo capitolo per ulteriori dettagli).

AGGIUNTE COMMODORE AL CP/M 3.0

Il Commodore ha aggiunto un numero di arricchimenti al CP/M 3.0. Questi arricchimenti adattano le capacità del Commodore 128 a quelle del CP/M 3.0. Includono cose come una linea di stato del dischetto visualizzata selettivamente, un disk drive virtuale, gestione locale/remota dei codici della tastiera, tasti programmabili di funzioni (stringhe) ed un numero di funzioni/caratteri ulteriori che sono assegnati a vari tasti. Questi arricchimenti sono descritti via via in questo capitolo.

FILE DEL CP/M

Ci sono due tipi di file del CP/M:

- I file **PROGRAM** o **COMMAND,** che consistono di una serie di istruzioni che il computer segue per ottenere un risultato specificato.
- I file **DATA,** che consistono di solito di un insieme di informazioni connesse (es. una lista di nomi di clienti e di indirizzi; inventari; registri di contabilità; il testo di un documento).

SPECIFICAZIONI DEI FILE

Un file CP/M viene identificato da una specificazione del file che può consistere di elementi individuali fino ad un numero di quattro, come segue:

- **Specificatore del Drive** (facoltativo), formato da una lettera singola seguita da due punti. Ad ogni drive è assegnata una lettera, nell'intervallo da A ad E. (E denota un drive virtuale, come spiegato più avanti nel capitolo).
- **Nome del file** (obbligatorio), che può essere lungo da uno ad otto caratteri. (questo è l'unico elemento obbligatorio della specificazione del file).
- **Tipo di file** (facoltativo), formato da uno a tre caratteri. Deve essere separato dal nome del file da un punto.
- **Password** (facoltativa), che può essere da uno ad otto caratteri. Deve essere separata dal tipo di file (o nome del file, se non è incluso il tipo di file) da punto e virgola.

ESEMPIO:

La seguente specificazione del file contiene tutti i quattro possibili elementi, tutti separati dai simboli appropriati:

A:DOCUMENT.LAW;FIREBIRD

NUMERO DELL'UTILIZZATORE

Il CP/M 3.0 inoltre identifica tutti i file assegnando a ciascuno un numero di utilizzatore, che può andare da 0 a 15. Il CP/M 3.0 assegna il numero di utilizzatore ad un file quando il file viene creato. I numeri di utilizzatore vi permettono di separare i file in sedici gruppi di file.

Qualsiasi numero di utilizzatore diverso da 0 deve precedere lo specificatore del drive. L'utilizzatore 0 che è il numero dell'utente per default, non viene visualizzato nella richiesta.

Se un file risiede nell'utilizzatore 0 ed è segnato con un attributo del file del sistema, si può accedere a quel file da qualsiasi numero di utilizzatore. Altrimenti, un comando può accedere solo a quei file che hanno il numero di utilizzatore corrente.

CREAZIONE DI UN FILE

Ci sono parecchi modi per creare un file CP/M, incluso:

- L'uso dell'editor del testo CP/M (ED).
- L'uso del comando PIP per copiare e rinominare un file.
- L'uso di un programma come MAC (un programma assembler CP/M in linguaggio macchina), che crea file di output mentre porta a compimento file di input.

L'USO DI CARATTERI WILDCARD PER ACCEDERE A PIÙ DI UN FILE

Un *wildcard* è un carattere che può essere usato in un nome del file o in un tipo di file al posto di altri caratteri. Il CP/M 3.0 usa il punto di domanda (?) e l'asterisco (*) come wildcard. Un ? vuol dire qualsiasi carattere che può essere incontrato in quella posizione. Un * dice al CP/M di riempire il nome del file con caratteri ? come indicato. Una specificazione del file che contiene un wildcard può riferirsi a più di un file e perciò è detta specificazione ambigua del file.

CARATTERI RISERVATI

I caratteri nella Tabella 15-1 sono caratteri riservati del CP/M 3.0. Usateli solo come indicato.

CARATTERE	SIGNIFICATO
<$,!>[ ] **TAB,SPAZIO,** **RITORNO CARRELLO**	**delimitatori delle specificazioni dei file**
:	**delimitatore del drive nella specific. del file**
.	**delimitatore di tipo di file nella specific. del file**
;	**delimitatore della password nella specific. del file**
;	**delimitatore del commento all'inizio di una linea di comando**
***?**	**caratteri wildcard in una specific. ambigua del file**
<>&! | \ + -	**delimitatori della lista delle opzioni**
[ ]	**delimitatori della lista delle opzioni per opzioni globali e locali**
()	**delimitatori per modificatori multipli nelle parentesi quadrate per opzioni che hanno modificatori**
/$	**delimitatori delle opzioni in una linea di comando**

Tabella 15-1. Caratteri riservati del CP/M 3.0

TIPI DI FILE RISERVATI

I tipi di file definiti nella Tabella 15-2 sono riservati per l'uso del sistema.

TIPO DI FILE	SIGNIFICATO
ASM	**file di sorgente dell'assembler**
BAS	**programma sorgente BASIC**
COM	**programma Z80 o equivalente in linguaggio macchina**
HEX	**file di output da MAC (usato da HEXCOM)**
HLP	**file di messaggio HELP**
$$$	**file temporaneo**
PRN	**stampa il file da MAC o RMAC**
REL	**file di output da RMAC (usato da LINK)**
SUB	**lista di comandi da eseguire da SUBMIT**
SYM	**file di simbolo da MAC, RMAC o LINK**
SYS	**file del sistema**
RSX	**Estensione del Sistema Residente (un file caricato automaticamente da un file di comando mentre se ne ha bisogno)**

Tabella 15-2. Tipi di file riservati in CP/M 3.0

COMANDI DEL CP/M

Ci sono due tipi di comandi nel CP/M 3.0:

- Comandi incorporati, che identificano i programmi in memoria.
- Comandi di utility transitorie, che identificano file di programmi.

Il CP/M 3.0 ha sei comandi incorporati ed oltre venti comandi di utility transitorie. Le utility possono essere aggiunte comprando programmi di applicazione compatibili CP/M 3.0. Inoltre, i programmatori esperti possono scrivere utility che operano con il CP/M 3.0.

COMANDI INCORPORATI

I comandi incorporati sono inseriti nella memoria del computer quando il CP/M 3.0 è caricato e sono sempre disponibili all'uso, senza tenere conto di quale disco sia in quale drive. La Tabella 15-3 lista i comandi incorporati del CP/M 3.0 del Commodore 128.

Alcuni comandi incorporati hanno opzioni che richiedono supporto da una utility transitoria in relazione con essi. Il comando della utility transitoria in relazione ha lo stesso nome del comando incorporato ed ha un tipo di file COM.

COMANDO	FUNZIONE
DIR	visualizza i nomi di file di tutti i file del directory tranne quelli segnati con l'attributo SYS.
DIRSYS	visualizza i nomi di file dei file segnati con l'attributo SYS (sistema) nel directory.
ERASE	cancella un nome di file dal directory del dischetto e libera lo spazio di memorizz. occupato dal file.
RENAME	rinomina un file del dischetto
TYPE	visualizza i contenuti di un file ASCII (TEXT) nel vostro schermo.
USER	cambia il numero di utilizzatore.

Tabella 15-3. Comandi incorporati

COMANDI DELLE UTILITY TRANSITORIE

I comandi delle utility transitorie del CP/M 3.0 sono listati nella Tabella 15-4. Quando una parola chiave di un comando che identifica una utility transitoria viene inserita, il CP/M 3.0 carica il file del programma dal dischetto e passa a quel file ogni nome di file, dato o parametro specificati in coda al comando.

USO DEI CARATTERI DI CONTROLLO PER L'EDITING (EDITAZIONE) DELLA LINEA

La Tabella 15-5 lista i caratteri di controllo per l'editing della linea per il CP/M 3.0 sul Commodore 128.

NOME	FUNZIONE
DATE	pone o visualizza la data ed il tempo.
DEVICE	assegna le unità logiche del CP/M ad una o più unità fisiche, cambia il protocollo del driver dell'unità e baud rate, o pone la misura della console dello schermo.
DIR	visualizza il directory con i file e le loro caratteristiche.
DUMP	visualizza un file in ASCII e in formato esadecimale.
ED	crea e modifica i file ASCII.
ERASE	usato per cancellamento wildcard.
GENCOM	crea un file speciale COM con un file RSX attaccato.
GET	ottiene temporaneamente l'input della console da un file del dischetto piuttosto che dalla tastiera.
FORMAT	inizializza il dischetto in formato GCR per uso del CP/M.
HELP	visualizza informazioni sull'uso comandi del CP/M 3.0.
INITDIR	inizializza un directory del dischetto per permettere la stampa di tempo e data.
KEYFIG	permette la ridefinizione dei tasti.
PATCH	visualizza o installa routine di correzione del sistema CP/M.
PIP	copia i file e combina i file.
PUT	dirige temporaneamente l'output della stampante o della console ad un file del dischetto.
RENAME	cambia il nome di un file, o di un gruppo di file, usando i caratteri wildcard.
SAVE	salva un programma in memoria in un dischetto.
SET	pone le opzioni del file incluse le etichette del dischetto, gli attributi dei file, il tipo di stampa del tempo e della data e la protezione della password.
SETDEF	pone le opzioni del sistema inclusa la catena di ricerca del drive.
SHOW	visualizza le statistiche del dischetto e del drive.
SUBMIT	esegue automaticamente i comandi multipli.
TYPE	visualizza il contenuto dei file di testo (o di gruppi di file, se sono usati i caratteri wildcard) sullo schermo (e sulla stampante se si desidera).

Tabella 15-4. Comandi delle Utility transitorie.

CARATTERE	SIGNIFICATO
CTRL-A o CURSORE SHIFT-LEFT	muove il cursore un carattere a sinistra.
CTRL-B	muove il cursore all'inizio della linea di comando senza effetto sui contenuti della linea. Se il cursore è all'inizio, CTRL-B lo muove alla fine della linea.
CTRL-E	forza un ritorno fisico del carrello ma non invia la linea di comando al CP/M 3.0. Muove il cursore all'inizio della linea seguente senza cancellare l'input precedente.
CTRL-F o CURSORE DESTRO	muove il cursore di un carattere a destra.
CTRL-G	cancella il carattere nella posizione corrente del cursore. Il cursore non si muove. I caratteri a destra del cursore si spostano a sinistra di una posizione.
CTRL-H	cancella il carattere a sinistra del cursore e lo muove a sinistra di un carattere. I caratteri a destra del cursore si spostano a sinistra di una posizione.
CTRL-I	muove il cursore allo stop tab successivo. Gli stop tab vengono automaticamente posti a ogni ottava colonna. Ha lo stesso effetto del tasto TAB.
CTRL-J	invia la linea di comando al CP/M 3.0 e ritorna il cursore all'inizio di una nuova linea. Ha lo stesso effetto di RETURN o di CTRL-M.
CTRL-K	cancella fino alla fine della linea dal cursore.
CTRL-M	invia la linea di comando al CP/M 3.0 e ritorna il cursore all'inizio di una nuova linea. Ha lo stesso effetto di RETURN o di CTRL-J.
CTRL-R	ridigita la linea di comando. Pone un carattere # nella locazione del cursore corrente, muove il cursore nella linea seguente e ridigita qualsiasi comando parziale che avete digitato.
CTRL-U	cancella tutti i caratteri nella linea di comando, pone un carattere # nella posizione del cursore corrente e muove il cursore nella linea successiva. Tuttavia, potete usare CTRL-W per richiamare qualsiasi carattere che era a sinistra del cursore quando avete digitato CTRL-U.
CTRL-W o ↑ CRSR ↓	richiama e visualizza le linee di comando inserite precedentemente sia a livello del sistema operativo che nei programmi esecutivi, se CTRL-W è il primo carattere inserito dopo la richiesta. CTRL-J, CTRL-M, CTRL-U e RETURN definiscono la linea di comando che potete richiamare. Se la linea di comando contiene dei caratteri, CTRL-W muove il cursore alla fine della linea di comando. Se digitate RETURN, il CP/M 3.0 esegue il comando richiamato.
CTRL-X	cancella tutti i caratteri a sinistra del cursore e muove il cursore all'inizio della linea corrente. CTRL-X salva qualsiasi carattere a destra del cursore.

Tabella 15-5. Caratteri di controllo della linea di Editing.

COME FARE DELLE COPIE DEI DISCHETTI E DEI FILE DEL CP/M 3.0

NOTA: Il comando **COPYSYS** del Digital Research Inc., usato in molti sistemi CP/M nella formattazione di un dischetto, non è implementato sul Commodore 128. Invece, il Commodore 128 usa uno speciale comando **FORMAT.**

Per fare salvataggi del dischetto del sistema CP/M, usate i programmi di utility FORMAT e PIP. FORMAT formatta il dischetto sia come un dischetto C128 a singola faccia che come un C128 a doppia faccia.

FARE COPIE CON UN SINGOLO DISK DRIVE

Potete copiare i contenuti di un dischetto con un singolo disk drive Commodore (1541 o 1571). Per esempio, usate la seguente sequenza di comandi per creare un dischetto del sistema CP/M che può subire il boot. Prima digitate:

```
A>FORMAT
```

e seguite le istruzioni date sullo schermo. Quando la copia del dischetto è formattato, digitate:

```
A>PIP E:=A:CPM+.SYS
```

Quando il file CPM + SYS è copiato, digitate:

```
A>PIP E:=A:CCP.COM
```

Per copiare tutto su un dischetto, usate la seguente sequenza di comandi:

```
A>FORMAT
A>PIP E:=A:*.*
```

Il sistema richiede all'utente di cambiare i dischetti come richiesto. Usate il drive A come drive di origine ed il drive E come drive di arrivo. Ci si riferisce al drive E come drive virtuale; cioè non esiste come pezzo di hardware reale, è strettamente un drive logico.

FARE COPIE CON DUE DISK DRIVE

Potete salvare i dischetti in CP/M usando due drive: il drive A ed il drive B. I drive possono essere etichettati con lettere diverse da A fino a D. Per fare delle copie dei vostro dischetto del sistema CP/M 3.0, usate prima l'utility FORMAT per copiare il caricatore del sistema operativo. Assicuratevi che il dischetto del sistema CP/M sia nel drive A, il drive di default e che il dischetto vuoto sia nel drive B. Poi inviate il seguente comando alla richiesta del sistema:

```
A>PIP B:=A:CPM+.SYS
```

Quando avete copiato il file CPM+.SYS, usate il comando PIP per copiare il file CCP.COM. Questo fornisce solo una copia del sistema operativo. Per copiare tutti i file dal dischetto del sistema, date il seguente comando PIP:

```
A>PIP B:=A:*.*
```

Questo comando PIP copia tutti i file sul dischetto del drive A nel dischetto del drive B. PIP visualizza il messaggio **COPYING,** seguito da ogni nome del file come procede l'operazione di copiatura. Quando PIP finisce di copiare, viene visualizzata la frase del sistema (A>).

SCHEMA GENERALE DEL SISTEMA CP/M 3.0

Il computer Commodore 128 è un sistema a due processori, con l'8502 come processore principale e lo Z80 come processore secondario. L'8502 ha lo stesso insieme di istruzioni del 6502. La funzione primaria dello Z80 è di lanciare il CP/M 3.0. Questa sezione descrive le esigenze ed i metodi generali per implementare il CP/M 3.0 sul C128. Quando il CP/M è lanciato, le normali funzioni del C128 non sono sopportate (il CP/M ed il BASIC non possono procedere allo stesso tempo). Il CP/M non sopporta direttamente nemmeno tutti i modi video del chip VIC. (Un'applicazione può essere scritta per procedere con il CP/M che può usare capacità grafiche ulteriori, ma l'applicazione dovrebbe dover tenere conto di tutti i dettagli, come le mappe della memoria).

COMPONENTI DEL SISTEMA OPERATIVO CP/M 3.0

Il sistema operativo del CP/M 3.0 è formato dai tre moduli seguenti: il Processore della Console di Comando (CCP), il Dischetto del Sistema Operativo in BASIC

(BDOS) ed il Sistema Basic di Input Output (BIOS). Il CCP è un programma che fornisce l'interfaccia basic dell'utente alle attrezzature del sistema operativo. Il CCP fornisce i sei comandi incorporati: **DIR, DIRS, ERASE, RENAME, TYPE** e **USER.** Il CCP opera nell'Area Provvisoria del Programma (TPA), la regione della memoria dove tutti i programmi di applicazione operano. Il CCP contiene il Modulo Caricatore del Programma, che carica i programmi (di applicazione) transitori dal dischetto nel TPA per l'esecuzione. Sul Commodore 128 un'area da 58K a 59K è fornita per il CP/M.

Il BDOS è nucleo logico e sistema del file del CP/M 3.0. Il BDOS fornisce l'interfaccia tra il programma di applicazione e le routine di input/output fisico del BIOS.

Il BIOS è un modulo dipendente dall'hardware che interfaccia il BDOS con un particolare ambiente hardware. Il BIOS porta a termine tutti gli I/O fisici del sistema. Il BIOS consiste di un numero di routine che sono configurate per sostenere l'hardware specifico del sistema del computer di obiettivo (in questo caso, il Commodore 128).

I moduli BDOS e BIOS cooperano per fornire al CCP e ad altri programmi transitori accessi alle attrezzature del CP/M 3.0 indipendenti dall'hardware. Piochè il BIOS può essere configurato per diversi ambienti hardware ed il BIOS rimane costante, potete trasferire programmi che operano con il CP/M 3.0 a sistemi con diverse configurazioni hardware.

COMUNICAZIONE TRA MODULI

Il BIOS carica il CCP nel TPA a sistema a freddo e partenze a caldo. Il CCP muove il Modulo Caricatore del Programma in cima al TPA ed usa il Modulo Caricatore del Programma per caricare programmi transitori.

Il BDOS contiene un insieme di funzioni che il CCP ed i programmi di applicazione chiamano per eseguire operazioni di input ed output di dischetto e di caratteri.

Il BIOS contiene una Tabella di Salto con un insieme di 33 punti di entrata che il BDOS chiama per eseguire funzioni primitive dipendenti dall'hardware, come I/O delle unità periferiche. Per esempio, **CONIN** è un punto di entrata del BIOS chiamato dal BDOS per leggere il carattere di input della console successiva.

Esistono delle somiglianze tra le funzioni BDOS e quelle BIOS, particolarmente per unità di I/O semplici. Per esempio, quando un programma transitorio chiama una funzione di output della console nel BDOS, il BDOS chiama un output della console nel BIOS. Nel caso di I/O del dischetto, tuttavia, il rapporto è più complesso. Il file BDOS può chiamare molte funzioni BIOS per eseguire una sola funzione di I/O del file BDOS. L'I/O del dischetto BDOS è in termini di 128 byte di record logici. L'I/O del dischetto BIOS è in termini di settori fisici e di tracce.

Il Blocco del Controllo del Sistema (SCB) è una struttura di dati CP/M 3.0 a 100 byte (decimali) che risiede nella componente del sistema BDOS. Il BDOS ed il BIOS comunicano per mezzo di campi nell'SCB. L'SCB contiene flag e dati BDOS, flag e dati CCP ed altre informazioni del sistema, come le caratteristiche della console ed il tempo e data correnti. Potete accedere ad alcuni campi del Blocco di Controllo del Sistema dal BIOS.

Notate che SCB contiene parametri critici del sistema che riflettono lo stato

corrente del sistema operativo. Se un programma modifica questi parametri, il sistema operativo si può rovinare. Consultate la Sezione 3 della *Guida al Sistema CP/M Plus* del DRI e la descrizione della Funzione 49 del BDOS nella *Guida del Programmatore del CP/M Plus* DRI per ulteriori informazioni sul Blocco del Controllo del Sistema.

La pagina zero è una regione della memoria che agisce da interfaccia tra i programmi transitori ed il sistema operativo. La pagina zero contiene parametri critici del sistema, inclusa l'entrata nel BDOS e l'entrata nella routine BOOT Calda del BIOS. Alla partenza del sistema, il BIOS inizializza questi due punti di entrata alla pagina zero. Tutta la correlazione tra i programmi transitori ed il BDOS è ristretta alla correlazione indiretta attraverso la pagina zero.

SINTESI DEL BIOS DEL CP/M 3.0

Questa sezione descrive l'organizzazione del BIOS CP/M 3.0 e del vettore di salto del BIOS. Fornisce una sintesi del Blocco di Controllo del Sistema, seguito da una trattazione delle procedure di inizializzazione del sistema, dell'I/O del carattere, del sostegno del clock, dell'I/O del dischetto e delle selezioni e movimenti della memoria.

ORGANIZZAZIONE DEL BIOS

Il BIOS è il modulo del CP/M 3.0 che contiene tutte le routine di input ed output dipendenti dall'hardware. Per configurare il CP/M 3.0 per un ambiente particolare hardware, il campione BIOS fornito della *Guida del Sistema CP/M Plus* deve essere adattato all'hardware specifico del sistema di obiettivo (il Commodore 128).

Il BIOS è essenzialmente un insieme di routine che esegue l'inizializzazione del sistema, l'I/O del carattere orientato verso la console e la stampante e l'I/O del settore fisico alle unità del dischetto. Il BIOS contiene anche routine che gestiscono movimenti del blocco e selezioni della memoria per la memoria a banco selettivo. Il BIOS fornisce tabelle che definiscono lo schema delle unità del dischetto e che assegnano lo spazio del buffer che il BDOS usa per raggruppare e dividere i record. Il BIOS può mantenere il tempo e la data del sistema nel Blocco di Controllo del Sistema.

La Tabella 15-6 descrive i punti di entrata nel BIOS dal Caricatore a partenza a freddo e dal BDOS. L'entrata al BIOS avviene attraverso un insieme di vettori di salto. La tabella di salto è un insieme di 33 istruzioni di salto che passano il controllo del programma alle subroutine individuali del BIOS.

N.	ISTRUZIONE	DESCRIZIONE
0	JMP BOOT	Esegue un'inizializz. di partenza a freddo.
1	JMP WBOOT	Esegue un'inizializz. di partenza a caldo.
2	JMP CONST	Controlla se è pronto il carattere di input della console.
3	JMP CONIN	Introduce il carattere della console.
4	JMP CONOUT	Registra in uscita il carattere della console.
5	JMP LIST	Registra in uscita il carattere della lista.
6	JMP AUXOUT	Scrive caratteri di output ausiliari.
7	JMP AUXIN	Legge caratteri di input ausiliari.
8	JMP HOME	Muove alla Traccia 00 sul dischetto selezionato.
9	JMP SELDSK	Seleziona il disk drive.
10	JMP SETTRK	Pone il numero della traccia.
11	JMP SETSEC	Pone il numero del settore.
12	JMP SETDMA	Pone l'indirizzo DMA.
13	JMP READ	Legge il settore specificato.
14	JMP WRITE	Scrive il settore specificato.
15	JMP LISTST	Riporta lo stato della lista.
16	JMP SECTRN	Traduce il settore logico in fisico.
17	JMP CONOST	Riporta lo stato di output della console.
18	JMP AUXIST	Riporta lo stato di input della porta aus.
19	JMP AUXOST	Riporta lo stato di output della porta aus.
20	JMP DEVTBL	Riporta l'indir. della tabella I/O del car.
21	JMP DEVINI	Inizial. le unità del car. di I/O.
22	JMP DRVTBL	Riporta l'indir. della tabella del disk drive.
23	JMP MULTIO	Pone i numeri dei settori logicamente consecutivi perchè siano letti o scritti.
24	JMP FLUSH	Forza lo scorrimento del buffer fisico per la divisione dei record sostenuta dall'utente.
25	JMP MOVE	Muove dalla memoria alla memoria.
26	JMP TIME	Pone/ottiene il segnale del tempo.
27	JMP SELMEM	Seleziona il banco della memoria.
28	JMP SETBNK	Specifica i banchi per il MOVE tra banchi.
29	JMP XMOVE	Specifica i banchi per un MOVE tra banchi.
30	JMP USERF	Funzioni del sistema CP/M Commodore.
31	JMP RESERV1	Riservato per un futuro uso.
32	JMP RESERV2	Riservato per un futuro uso.

Tabella 15-6. Vettori di Salto BIOS del CP/M 3.0.

Tutti i punti di entrata nel vettore di salto BIOS sono inclusi nel BIOS CP/M 3.0 del C128.

Ogni indirizzo di salto nella Tabella 15-6 corrisponde ad una particolare subroutine che esegue una specifica operazione del sistema. Notate che due punti di entrata sono riservati per versioni future del CP/M ed un punto di entrata è dato per le funzioni del sistema Commodore.

La Tabella 15-7 mostra le cinque categorie delle operazioni di sistema e le chiamate delle funzioni che compiono queste operazioni.

OPERAZIONE	FUNZIONE
Inizializz. del Sistema	**BOOT,WBOOT,DEVTBL,DEVINI,DRVTBL**
Carattere I/O	**CONST,CONIN,CONOUT,LIST,AUXOUT, AUXIN,LISTST, CONOST,AUXIST,AUXOST**
I/O del Dischetto	**HOME,SELDSK,SETTRK,SETSEC, SETDMA,READ,WRITE, SECTRN,MULTIO,FLUSH**
Selezioni e Movimenti di Memoria	**MOVE,SELMEM,SETBNK,XMOVE**
Supporto del Clock	**TIME**

Tabella 15-7. Funzioni BIOS del CP/M 3.0.

L'appendice K illustra come chiamare una funzione di sistema del CP/M Commodore in linguaggio macchina Z80.

ORGANIZZAZIONE DELLA MEMORIA DEL SISTEMA

La Figura 15-1 mostra l'organizzazione generale della memoria del CP/M 3.0.

La mappa della memoria è limitata a 64K per qualsiasi singolo punto per volta. Tuttavia, il banco della RAM può essere selezionato e le diverse aree della ROM possono coprire la RAM (con riempimenti durante le operazioni di scrittura). L'attuale mappa della memoria è controllata dall'MMU. Si può accedere all'MMU nell'area di I/O (solo quando l'I/O è abilitato nello spazio Z80) o con il Registro della Configurazione di Caricamento localizzato da FF00 a FF04.

Se i Registri della Configurazione di Caricamento vengono letti, allora viene letto il valore corrente. Una scrittura in FF00 cambia la configurazione dopo aver completato l'istruzione corrente. Una scrittura da FF01 a FF04 aggiorna la configurazione corrente nel valore memorizzato nei Registri di Preconfigurazione (il dato scritto non è usato). I puntatori delle pagine MMU hanno entrambi un puntatore (di pagina) basso ed alto. Quello alto viene scritto per primo e tenuto nell'MMU; il valore alto viene aggiornato dal registro tampone (latch) quando viene scritto il byte basso. I Registri di Controllo MMU sono listati nel Capitolo 14.

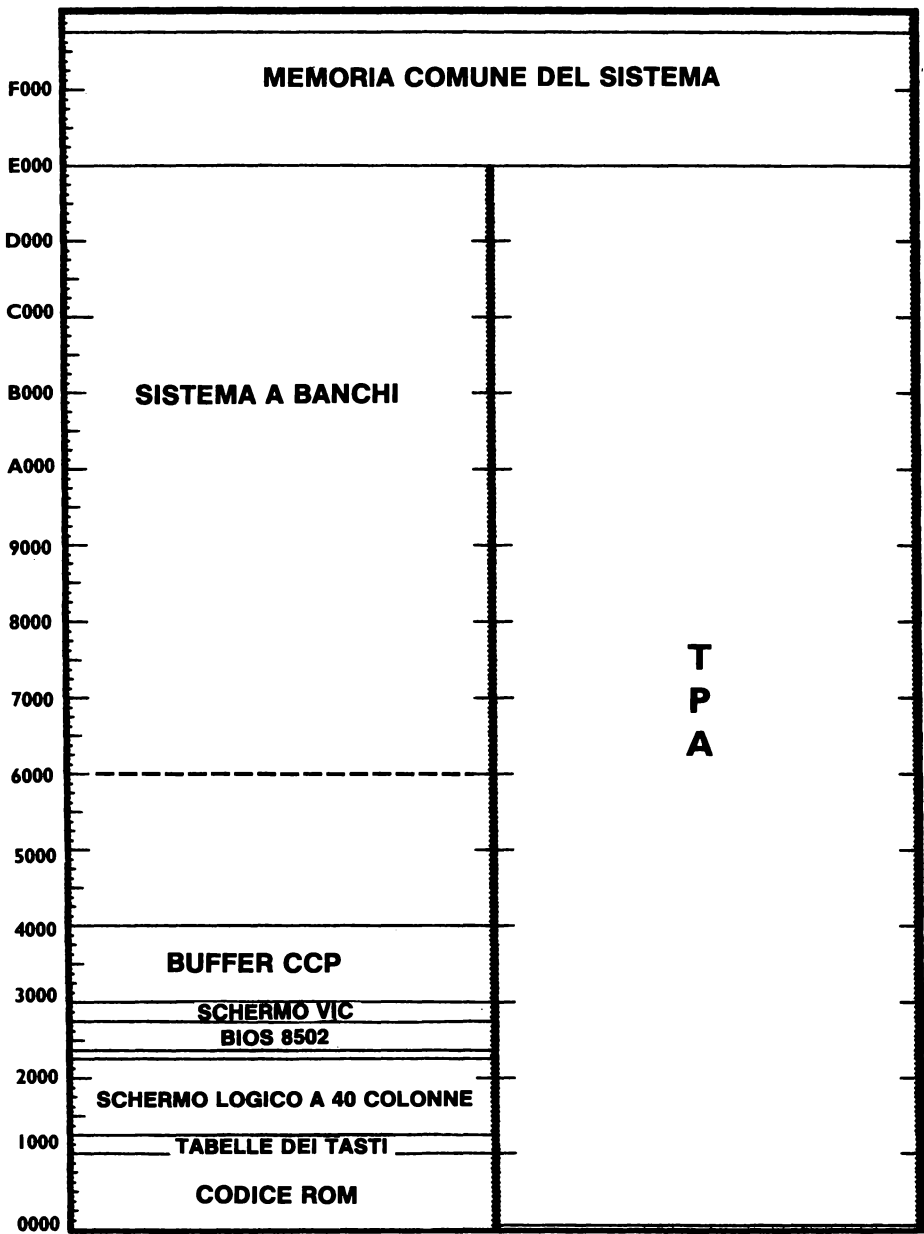



Figura 15-1. Organizzazione generale della Memoria del CP/M 3.0

ORGANIZZAZIONE DEL DISCHETTO

Il CP/M 3.0 fornisce un numero di diversi formati di dischetto, inclusi tre formati Commodore ed un numero di formati MFM. (MFM è il formato standard industriale).

Il primo formato Commodore è quello GCR Commodore a singola faccia, che è compatibile con il CP/M 2.2 che va sul Commodore 64. Con questo formato, il Blocco di Controllo dei File (FCB) viene inserito come 32 tracce di 17 settori ciascuna ed un offset della traccia di 2. La routine BIOS aggiunge un 1 alle tracce maggiori di 18 (questa è la traccia del directory C64).

Il secondo formato, conosciuto come il formato del dischetto CP/M Plus del C128, è nuovo ed è anche a singola faccia. Questo formato, che è anche un formato GCR, si avvantaggia della capacità dell'intero dischetto predisponendo l'FCB con 638 tracce di un settore ciascuna ed un offset della traccia di 0. Questo ha l'effetto di fare si che il CP/M ponga la traccia col numero del blocco relativo all'inizio del dischetto, con il settore posto sempre a 0. L'algoritmo seguente è usato per convertire la TRACCIA richiesta in un numero reale di traccia e di settore.

TRACCIA RICHIESTA	TRACCIA REALE
000 >= TRACCIA>355	**((TRACCIA+2)/21)+1**
355 >= TRACCIA>487	**((TRACCIA-354)/19)+18**
487 >= TRACCIA>595	**((TRACCIA-487)/18)+25**
595 >= TRACCIA>680	**((TRACCIA-595)/17)+31**
680>= TRACCIA>1360	**PONE LATO 2 DELLA TRACCIA=TRACCIA-680**

Il settore effettivo viene tradotto per fornire un disallineamento che rende più veloci le operazioni. Il disallineamento viene usato solo con il nuovo formato più grande. Viene usata una diversa tabella di disallineamento per ogni regione del dischetto.

Il terzo formato Commodore è un formato GCR a doppia faccia. Il dischetto viene trattato come avente 1276 settori di dati con un offset della traccia di 0. La faccia 1 viene usata per prima; poi viene usata la faccia 2.

NOTA: Questo non è il metodo usuale di trattamento di un dischetto a doppia faccia; tuttavia, sistemando il dischetto in questo modo, l'utilizzatore del 1541 sarà ancora in grado di leggere i dati scritti all'inizio di un dischetto a doppia faccia.

Il terzo formato Commodore e tutti i formati MFM richiedono che l'utente abbia il nuovo drive 1571. Questo drive sopporta entrambi i dischetti a singola e a doppia faccia e sia il formato di codificazione dei dati Commodore GCR che quello industriale standard MFM.

La tabella seguente riassume le capacità dei drive 1541/1571 tenendo conto dei vari formati dei dischetti.

FORMATO DEL DISCHETTO	DRIVE 1541	DRIVE 1571
C64 GCR a singola faccia	√	√
C128 GCR a singola faccia	√	√
C128 GCR a doppia faccia		√
MFM Formato		√

FORMATO DEL DISCHETTO CP/M C64 (A SINGOLA FACCIA)

Questo formato, mostrato nella Figura 15-2, viene fornito per permettere all'utente di leggere/scrivere file creati usando la cartuccia CP/M 2.2 del C64. (Tuttavia, non usate la cartuccia CP/M 2.2 con il C128). Notate lo spazio non utilizzato di questo formato.

FORMATI DEL DISCHETTO CP/M DEL C128 (A SINGOLA E DOPPIA FACCIA)

La Figura 15-3 mostra il formato CP/M Plus del C128 per un dischetto a singola faccia.

NOTA: Questo formato viene duplicato sulla seconda faccia di un dischetto a doppia faccia (con l'eccezione di B, che rimane non utilizzato e perciò diventa X).

SETTORE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	x	x	x	x
1	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	x	x	x	x
2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
6	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
7	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
8	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
9	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
10	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
11	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
12	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
13	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
14	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
15	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x
17	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D		
18	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x		
19	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x		
20	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x		
21	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x		
22	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x		
23	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	x		
24	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x			
25	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x			
26	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x			
27	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x			
28	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x			
29	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.				
30	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.				
31	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.				
32	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.				
33	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.				
34	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.				

. Usato dal CP/M.
B Settore del Boot (Sistema).
D Settore del Directory (Dischetto DOS).
x Non usato dal CP/M.

Figura 15-2. Formato del Dischetto C64 CP/M

SETTORE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	B	.	.	.	.	x	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
5	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
6	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
7	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
8	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
9	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
10	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
11	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
12	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
13	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
14	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
15	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
17	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		
18	D	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		
19	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		
20	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		
21	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		
22	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		
23	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		
24	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.			
25	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.			
26	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.			
27	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.			
28	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.			
29	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.			
30	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.				
31	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.				
32	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.				
33	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.				
34	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.				

. Usato dal CP/M,reg.=regione (1-4).
B Settore del Boot (Sistema).
D Settore del Directory (Dischetto DOS).
x Non Usato dal CP/M.

TABELLA DI DISALLINEAMENTO

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	00	17	13	09	05	01	18	14	10	06	02	19	15	11	07	03	20	16	12	08	04
2	00	04	08	12	16	01	05	09	13	17	02	06	10	14	18	03	07	11	15		
3	00	11	04	15	08	01	12	05	16	09	02	13	06	17	10	03	14	07			
4	00	07	14	04	11	01	08	15	05	12	02	09	16	06	13	03	10				

Figura 15-3. Formato del Dischetto C128 CP/M Plus

FORMATI DEL DISCHETTO MFM

Un numero di formati del dischetto MFM vengono inseriti nel CP/M del C128. Questi formati, che possono essere selezionati nel momento in cui si vuole far girare un programma, includono Osborne, Kaypro, Epson ed il CP/M-86 IBM. La capacità del CP/M-86 IBM viene fornita in modo che i dati possano essere trasferiti tra le macchine. Tuttavia i programmi CP/M-86 non possono girare sul C128, poichè sul C128 il CP/M Plus gira sullo Z80, non su un 8088.

Quando è usato con il Disk Drive 1571, il Commodore 128 fornisce una varietà di formati di dischetti MFM a doppia densità (per la lettura e/o la scrittura), inclusi:

Epson QX10	(settori di 512 byte, doppia faccia, 10 settori per traccia)
IBM-8 SS (CP/M 86)	(settori di 512 byte, singola faccia, 8 settori per traccia)
IBM-8 DS (CP/M 86)	(settori di 512 byte, doppia faccia, 8 settori per traccia)
KayPro II	(settori di 512 byte, singola faccia, 10 settori per traccia)
KayPro IV	(settori di 512 byte, doppia faccia, 10 settori per traccia)
Osborne DD	(settori di 1024 byte, singola faccia, 5 settori per traccia)

Quando inserite uno di questi dischetti nel drive e tentate di accedervi, il sistema sente il tipo di dischetto in relazione al numero di byte per settore ed al numero di settori per traccia. Se il formato del dischetto non è unico, viene visualizzato un rettangolino nell'angolo in basso a sinistra dello schermo, mostrandovi a che tipo di dischetto state accedendo. Il sistema vi richiede di selezionare il tipo specifico di dischetto facendo lo scrolling delle possibilità date in questa finestra.

NOTA: Le possibilità di scelta vengono date una alla volta; passatele usando i tasti delle frecce a destra ed a sinistra. Premete **RETURN** quando il tipo di dischetto che sapete che è nel drive viene visualizzato. Premete **CONTROL RETURN** per bloccare questo formato del dischetto così che non dovrete selezionare il tipo di dischetto tutte le volte che accedete al drive.

ANALISI DELLA TASTIERA

La routine di analisi della tastiera che viene chiamata per ottenere un carattere della tastiera riporta il codice del tasto premuto, o un codice che indica che nessun tasto al momento è premuto. Il codice di analisi della tastiera è responsabile anche della gestione dei tasti programmabili, tasti di funzioni programmabili, della disposizione dei caratteri e dei colori di fondo, della selezione dei formati del dischetto MFM e del tipo di emulazione dello schermo corrente.

Qualsiasi tasto della tastiera può essere definito per generare un codice o una funzione tranne i tasti seguenti:

LEFT SHIFT
RIGHT SHIFT
SHIFT LOCK
TASTO COMMODORE
CONTROL
RESTORE (8502 NMI)
40/80 DISPLAY
CAPS LOCK KEY

La tastiera riconosce le seguenti funzioni speciali:

Tasto cursore a sinistra	-Usato per definire un tasto
Tasto cursore a destra	-Usato per definire una stringa (punta ai tasti delle funzioni)
Tasto ALT	-Usato come filtro a levetta del tasto

Per indicare queste funzioni, tenete premuto il tasto **CONTROL** ed il tasto **RIGHT SHIFT** e premete allo stesso tempo il tasto della funzione desiderata.

DEFINIZIONE DI UN TASTO

Il programma di utility **KEYFIG** permette all'utente di definire il codice che un tasto può produrre. Ogni tasto ha quattro modi d'uso per questa funzione:

- Normale
- Scorrimento in ordine alfabetico
- Scorrimento
- Controllo

Il modo scorrimento in ordine alfabetico viene inserito/disinserito premendo il tasto Commodore. Quando questo modo è inserito, appare un piccolo rettangolino bianco nella parte bassa dello schermo. Il primo tasto che viene digitato dopo questa apparizione è il tasto che deve essere definito. Viene visualizzato il valore esadecimale corrente assegnato a questo tasto e l'utente può digitare il nuovo codice esadecimale del tasto o distruggerlo digitando un tasto non esadecimale. Quella che segue è una definizione dei codici che possono essere assegnati ad un tasto. Per ulteriori informazioni consultate KEYFIG HELP.

CODICE	**FUNZIONE**
00h	**Zero (la stessa come se il tasto non fosse premuto)**
da 01h a 7Fh	**Codici ASCII normali**
da 80h a 9Fh	**Stringa assegnata**
da A0h a AFh	**Colore con carattere ad 80 colonne**
da B0h a BFh	**Colore di fondo ad 80 colonne**
da C0h a CFh	**Colore con carattere a 40 colonne**
da D0h a DFh	**Colore di fondo a 40 colonne**
da E0h a EFh	**Colore di cornice a 40 colonne**
F0h	**levetta dello stato del dischetto inserita/disinserita**
F1h	**Pausa del Sistema**
F2h	**(Non definito)**
F3h	**Finestra dello schermo a 40 colonne a destra**
F4h	**Finestra dello schermo a 40 colonne a sinistra**
da F5h a FFh	**(Non definito)**

DEFINIZIONE DI UNA STRINGA

La funzione **DEFINE STRING** permette all'utente di assegnare più di un codice di tasto ad un singolo tasto. Qualsiasi tasto che venga digitato in questo modo viene posto nella stringa. L'utente può vedere i risultati della digitazione in un lungo rettangolo in fondo allo schermo. Notate, tuttavia, che alcuni tasti possono non visualizzare quello che sono.

Per permettere all'utente di controllare l'inserimento dei dati, vengono fornite cinque speciali funzioni di editing. Per accedere ad una qualsiasi di queste funzioni, premete prima **CONTROL** e poi **RIGHT SHIFT.** (Questo permette all'utente di inserire qualsiasi dato nel buffer). Le funzioni assegnate alle cinque stringhe dei tasti edit (di editing) sono le seguenti:

TASTO	FUNZIONE
RETURN	**Termina la definizione della stringa.**
simbolo + *	**Inserisce lo spazio nella stringa.**
simbolo – *	**Cancella il carattere cursore.**
Cursore destro	**Muove il cursore a destra.**
Cursore sinistro	**Muove il cursore a sinistra.**

* Solo sulla tastiera principale.

MODO ALT

Questa funzione è una levetta (accesa/spenta) e viene data per permettere all'utente di inviare i codici ad 8 bit ad un'applicazione senza che il circuito della tastiera "mangi" il codice da 80h a FFh.

AGGIORNAMENTO DEL VIDEO 40/80 COLONNE

Come notato altrove in questo libro, ci sono due diversi sistemi di visualizzazione nel C128. Il primo, che è controllato dal chip VIC, produce un video di 25 linee per 40 colonne, ha molti modi grafici di operazione e può essere usato con una televisione a colori standard (o in bianco e nero) o con un monitor a colori. (Consultate i Capitoli 9 e 10 per i dettagli). Il solo modo video controllato dal VIC e usato dal CP/M è il modo carattere standard, ogni sfondo dei carattere e dello schermo ha fino a sedici colori.

Il secondo sistema di visualizzazione disponibile nel CP/M del C128 è controllato dal controllore video 8563. Il formato video di questo controllore è di 25 linee per 80 colonne, con attributi colori del carattere. Il chip VIC è un video con memoria a mappa e l'8563 è controllato per l'I/O. I due sottosistemi video sono trattati come due video separati. Il CP/M 3.0 può assegnarne uno o entrambi all'unità console di output.

Entrambi i video sono controllati da un pacchetto comune di emulazione di un terminale, un circuito Lear Siegler ADM-31 (ADM-3A è un sottoinsieme di questo). Il circuito terminale è diviso in due parti: Emulazione del terminale e funzioni del terminale. L'emulazione del terminale è gestita dal BIOS dello Z80 e la funzione del terminale è gestita principalmente nella ROM Z80.

La sezione seguente mostra i vari protocolli dell'emulazione del terminale sopportati dal CP/M del Commodore 128.

PROTOCOLLI DELL'EMULAZIONE DEL TERMINALE

FUNZIONE	**SEQUENZA DEL CARATTERE**	**CODICE DEL CARATTERE ESADEC.**
Posizione cursore	**ESC(riga # +32) (col # +32)**	**1B 3D 20+20**
Cursore a sinistra	**Control H**	**08**
Cursore a destra	**Control L**	**0C**
Cursore giù	**Control J**	**0A**
Cursore su	**Control K**	**0B**
Home e Azzera Schermo	**Control Z**	**1A**
Ritorno Carrello	**Control M**	**0D**
Escape	**Control [**	**1B**
Campanello	**Control G**	**07**

NOTA: Il video è 24 (1-24 per 80 (1-80): l'origine del cursore è sempre 1/1.

Figura 15-4. Protocollo Lear Siegler ADM-3A.

NOTA: Le seguenti sono state aggiunte per permettere al sistema di emulare il video KayPro II più da vicino.

FUNZIONE	SEQUENZA DEI CARATTERI
Cursore Principale	Control ↑
CEL (Azzera fino alla Fine della linea)	Control-X
CES (Azzera fino alla Fine dello Schermo)	Control-W

	SEQUENZA DEL CARATTERE	CODICI DEL CARATTERE ESADEC.
Azzera fino alla fine della linea	ESC T	1B 54
	ESC t	1B 74
Azzera fino alla fine dello schermo	ESC Y	1B 59
	ESC y	1B 79
Cursore home (principale)	ESC:	1B 3A
e azzera schermo	ESC *	1B 2A
Metà intensità on	ESC)	1B 29
Metà intensità off	ESC (	1B 28
Video inverso on	ESC G4	1B 47 34
Lampeggio on	ESC G2	1B 47 32
Sottolineatura on *	ESC G3	1B 47 33
Seleziona il carattere alternato	ESC G1	1B 47 31
Video inverso e lampeggio off	ESC G0	1B 47 30
Inserisci linea	ESC E	1B 45
Inserisci carattere	ESC Q	1B 51
Cancella linea	ESC R	1B 52
Cancella carattere	ESC W	1B 57
Poni colori dello schermo	ESC ESC ESC colore #	

dove il colore # = da 20h a 2Fh

da 30h a 3Fh colore di fondo
da 40h a 4Fh colore del bordo
} colori fisici

da 50h a 50Fh colore del carattere
da 60h a 60Fh colore di fondo
da 70h a 70Fh colore del bordo (solo 40 colonne)
} colori logici

* Indica che questa non è una normale sequenza ADM31.
Nota: Il video è 24(1-24) per 80(1-80); l'origine del cursore è sempre 1/1.

Figura 15-5. Protocollo Lear Siegler ADM-31

OPERAZIONI DEL SISTEMA

PREDISPOSIZIONE DELL'ORA DEL SISTEMA

L'ora del giorno è posta con questa funzione. L'ora del giorno viene memorizzata nel formato impaccato BCD nel Blocco di Controllo del Sistema (SCB) in tre locazioni (ore, minuti, secondi). Questa routine legge il tempo SCB e lo scrive nel clock dell'ora del giorno all'interno del 6526. Questo tempo viene aggiornato nel chip ed usato dal CP/M. Lo Z80 è in grado di leggere/scrivere nel 6526 direttamente.

AGGIORNAMENTO DELL'ORA DEL SISTEMA

Il tempo SCB viene aggiornato dal clock dell'ora del giorno nel 6526 eseguendo una chiamata del sistema.

ORGANIZZAZIONE DEL BIOS 8502

L'8502 è responsabile della maggior parte delle funzioni di I/O a basso livello. La richiesta di queste funzioni viene fatta attraverso una serie zone di memoria riservate (Mail box). Una volta che le mail box vengono predisposte, lo Z80 le chiude e l'8502 parte (BIOS85). L'8502 guarda i comandi nelle mail box ed esegue le richieste, pone lo stato di comando e chiude. Lo Z80 viene riabilitato; esso guarda lo stato del comando e esegue le azioni appropriate.

I comandi BIOS 8502 sono definiti nell'Appendice K.

Questo conclude la spiegazione del riassunto del sistema CP/M del Commodore 128. Tuttavia, il Sistema CP/M del Commodore 128 include molte routine e ulteriori funzioni dipendenti dal 128 che vengono eseguite dal microprocessore Z80. Poichè la maggior parte di queste routine, chiamate del sistema e funzioni sono informazioni tabulari, sono trattate nell'Appendice K. Per informazioni su qualsiasi dei seguenti argomenti, fate riferimento all'Appendice K.

Comandi BIOS 8502
Tutte le funzioni dell'utente dipendenti dal sistema Z80 del Commodore 128
Chiamata di un CP/M BIOS, BIOS 8502 e funzioni del sistema CP/M dell'utente in linguaggio macchina Z80
Più informazioni sui formati del dischetto MFM
Mappa della Memoria del CP/M del Commodore 128 (Z80)

16

MAPPE DELLA MEMORIA DEL COMMODORE 128 E DEL COMMODORE 64

Questo capitolo fornisce le mappe della memoria per entrambi i modi C128 e C64. Una mappa della memoria vi dice esattamente come è organizzata una memoria internamente sia nella RAM che nella ROM. Vi dice esattamente cosa risiede in ogni locazione di memoria, vi aiuta a trovare i vettori di indirizzo per le routine ed i punti di entrata e vi dà informazioni sullo schema generale del computer. La mappa della memoria è probabilmente l'accessorio più importante della programmazione. Fate riferimento alla mappa della memoria tutte le volte che avete bisogno di direzioni nella memoria del vostro C128. Gli indirizzi listati con più di un'etichetta di indirizzo vengono usati per più di uno scopo. In BASIC, la variabile ha uno scopo; nel Monitor di Linguaggio Macchina può averne un altro.

Le convenzioni usate per le mappe della memoria sono le seguenti:

Colonna1	Colonna2	Colonna3	Colonna4
ETICHETTA DI INDIRIZZO DI MEMORIA	INDIRIZZO ESADECIMALE	INDIRIZZO DECIMALE	DESCRIZIONE

Consultate l'Appendice K per la mappa di memoria Z80 per il CP/M sul Commodore 128.

MAPPA DELLA MEMORIA DEL C128

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
D6510	0000	0	REGISTRO DIREZ. DATI 6510
R6510	0001	1	REGISTRO DATI 6510
BANK	0002	2	SIMBOLO "SEARCH" CERCA O BANCO#
PC-HI	0003	3	INDIR. PER COMANDO O MONITOR DEL SIS. BASIC E ROUTINE LONG CALL/ JUMP
PC-LO	0004	4	INDIRIZZO, STATO, REG-A, REG-X, REG-Y
S-REG	0005	5	TEMP. REG. DI STATO
A-REG	0006	6	TEMP.REG.A
X-REG	0007	7	TEMP.REG.X
Y-REG	0008	8	TEMP.REG.Y
STKPTR	0009	9	TEMP. PUNTATORE DELLO STACK

MEMORIZZAZIONE BASIC DI PAGINA ZERO

INTEGR CHARAC	0009	9	CERCA IL CARATTERE

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
ENDCHR	000A	10	FLAG: ANALIZZ. RICHIESTA A FINE STRINGA
TRMPOS	000B	11	COLONNA DI SCHERMO DA ULTIMA TAB
VERCK	000C	12	FLAG:0= CARICA,1= VERIFICA
COUNT	000D	13	INPUT BUF. PTR/#DEI SUBSCRITTI
DIMFLG	000E	14	FLAG:DIMENS. ARRAY DI DEFAULT
VALTYP	000F	15	DIGITA DATO: $FF=STRINGA, $00=NUMERICO
INTFLG	0010	16	DIGITA DATO: $00=FLOAT.PT, $80=INTERO
GARBFL	0011	17	FLAG:ANAL. DATO/LISTA RICHIESTA/ INSIEME GARBAGE
DORES			
SUBFLG	0012	18	FLAG:RIF.A SUBSCRITTO/ CHIAMATA FUNZIONE DELL'UTENTE
INPFLG	0013	19	FLAG:$00= INPUT,$40= OTTIENI, $98 =LEGGI
DOMASK	0014	20	
TANSGN			FLAG:SEGNO TAN/COMPARA IL RISULTATO
CHANNL	0015	21	
POKER	0016	22	
LINNUM			VALORE DEL TEMP INTERO
TEMPPT	0018	24	PUNTATORE: STACK DELLA STRINGA DEL TEMP
LASTPT	0019	25	INDIR. ULTIMA STRINGA DEL TEMP
TEMPST	001B	27	STACK PER STRINGHE DEL TEMP
INDEX	0024	36	AREA PUNTATORE DI UTILITY
INDEX1			
INDEX2	0026	38	
RESHO	0028	40	FLOAT.PT. PRODOTTO DI MOLTIPLICAZ.
RESMOH	0029	41	
ADDEND	002A	42	
RESMO			
RESLO	002B	43	
TXTTAB	002D	45	PUNTATORE: INIZIO TESTO IN BASIC
VARTAB	002F	47	PUNTATORE: INIZIO VARIABILI BASIC
ARYTAB	0031	49	PUNTATORE: INIZIO ARRAY BASIC
STREND	0033	51	PUNTATORE: FINE ARRAY BASIC+1
FRETOP	0035	53	PUNTATORE: FONDO DELLA MEMORIZZAZ. STRINGHE
FRESPC	0037	55	PUNTATORE STRINGA UTILITY
MAX-MEM-1	0039	57	CIMA BANCO STRINGHE/VARIABILI (BANCO 1)
CURLIN	003B	59	NUM. LINEA BASIC CORRENTE

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
TXTPTR	003D	61	PUNTATORE TESTO BASIC USATO DA CHRGET,ECC.
FORM	003F	63	USATO DA PRINT CHE USA IL PUNTATO-
FNDPNT			RE DI ITEM TROVATO DA SEARCH
DATLIN	0041	65	NUMERO LINEA DATI CORRENTE
DATPTR	0043	67	INDIR. ITEM DATI CORRENTE
INPPTR	0045	69	VETTORE: ROUTINE DI INPUT
VARNAM	0047	71	NOME DELLA VARIABILE BASIC CORRENTE
FDECPT	0049	73	
VARPNT			PUNTATORE: VARIABILE DATI BASIC CORRENTE
LSTPNT	004B	75	
FORPNT			PUNTATORE: VARIABILE INDICE PER FOR/NEXT
ANDMSK			
EORMSK	004C	76	
VARTXT	004D	77	
OPPTR			
OPMASK	004F	79	
GRBPNT	0050	80	
TEMPF3			
DEFPNT			
DSCPNT	0052	82	
	0054	84	
HELPER	0055	85	FLAG:"HELP" O "LIST"
JMPER	0056	86	
	0057	87	
OLDOV	0058	88	
TEMPF1	0059	89	
PTARG1			MOLTIPLIC. DEFINITO PER L'ISTRUZIONE
PTARG2	005B	91	
STR1	005D	93	
STR2	0060	96	
POSITN	0063	99	
MATCH	0064	100	
ARYPNT	005A	90	
HIGHDS			
HIGHTR	005C	92	
TEMPF2	005E	94	
DECCNT	005F	95	NUMERO DI CIFRE DOPO IL PUNTO DECIMALE
TENEXP	0060	96	
T0			ML MONITOR Z.P. MEMORIZ. IN FAC
GRBTOP	0061	97	
DPTFLG			FLAG PUNTO DECIMALE

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
LOWTR			
EXPSGN	0062	98	
FAC	0063	99	
DSCTMP			
LEFT-FLAG			DISEGNA FLAG SINISTRO
FACEXP			FAC#1 ESPONENTE
T1			MONITOR Z.P. MEMORIZ.IN FAC
RIGHT-FLAG	0064	100	DISEGNA FLAG DESTRO
FACHO			FAC#1 MANTISSA
FACMOH	0065	101	
INDICE	0066	102	
FACMO			
T2			MONITOR Z.P. MEMORIZ. IN FAC
FACLO	0067	103	
FACSGN	0068	104	SEGNO FAC#1
DEGREE	0069	105	
SGNFLG			PUNTATORE: VAL. SERIE COSTANTE
ARGEXP	006A	106	ESPONENTE FAC#2
ARGHO	006B	107	MANTISSA FAC#2
ARGMOH	006C	108	
INIT-AS-0			SOLO UN CONTO PER INIT
ARGMO	006D	109	
ARGLO	006E	110	
ARGSGN	006F	111	SEGNO FAC#2
STRNG1	0070	112	
ARISGN			RISULTATO DI COMPARAZIONE DEL SEGNO:FAC #1 CONTRO #2
FACOV	0071	113	FAC#1 ORDINE BASSO (ARROTONDA-TO)
STRNG2	0072	114	
POLYPT			
CURTOL			
FBUFPT			PUNTATORE: BUFFER DELLA CASSETTA
AUTINC	0074	116	INC. VALORE PER AUTO (0=OFF)
MVDFLG	0076	118	FLAG SE 10K A RATE SISTEMATI
Z-P-TEMP-1	0077	119	STAMPA IL CONTATORE DELLO ZERO NON SIGNIFICATIVO MOVSPR E SPRITE TEMPORANEAM. MID$ TEMPORANEAM.
HULP	0078	120	CONTATORE
KEYSIZ			
SYNTMP	0079	121	USATO COME TEMP PER CARICAMENTI INDIRETTI
DSDESC	007A	122	DESCRITTORE PER DS$
TXTPTR			MEMORIZ. DEL MONITOR Z.P.
TOS	007D	125	ALTO DELLO STACK DEL TEMPO DI ESECUZIONE

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
RUNMOD	007F	127	FLAG ESECUZ./ MODO DIRETTO
PARSTS	0080	128	STATO DELLA PAROLA PARSER DOS
POINT			PUNTATORE D'USO IN DEC. PT
PARSTX	0081	129	
OLDSTK	0082	130	
MEMORIZZAZIONE Z-P PER COMANDI GRAFICI			
COLSEL	0083	131	SELEZIONATO IL COLORE CORRENTE
MULTICOLOR-1	0084	132	
MULTICOLOR-2	0085	133	
FOREGROUND	0086	134	
SCALE-X	0087	135	FATTORE SCALA IN X
SCALE-Y	0089	137	FATTORE SCALA IN Y
STOPNB	008B	139	STOP DISEGNO SE NON SFONDO/NO STESSO COLORE
GRAPNT	008C	140	
VTEMP1	008E	142	
VTEMP2	008F	143	
MEMORIZZAZIONE EDITOR/KERNAL			
STATUS	0090	144	BYTE DI I/O DELLO STATO DELL'OPERAZIONE
STKEY	0091	145	FLAG STOP TASTO
SVTX	0092	146	REGISTRA TEMPORANEAMENTE
VERCK	0093	147	CARICA O VERIFICA IL FLAG
C3PO	0094	148	FLAG DEL CAR SERIALE BUFFERIZZATO
BSOUR	0095	149	BUFFER DEL CAR PER SERIALE
SYNO	0096	150	SINC# DELLA CASSETTA
XSAV	0097	151	TEMP PER BASIN
LDTND	0098	152	INDICE DEL FILE LOGICO
DFLTN	0099	153	UNITÀ# DI INPUT PER DEFAULT
DFLTO	009A	154	UNITÀ# DI OUTPUT PER DEFAULT
PRTY	009B	155	PARITÀ DI CASSETTA
DPSW	009C	156	INTERRUTTORE DELLA CASSETTA DIPOLE
MSGFLG	009D	157	FLAG DEL MESSAGGIO OS
PTR1	009E	158	ERRORE CASSETTA PASSO 1
T1			TEMPORANEAM. 1
PTR2	009F	159	ERRORE CASSETTA PASSO 2
T2			TEMPORANEAM. 2
TIME	00A0	160	CLOCK 24 ORE IN 1/60ESIMO DI SECONDO
R2D2	00A3	163	USO BUS SERIALE
PCNTR			CASSETTA
BSOUR1	00A4	164	TEMP USATO DA ROUTINE SERIALE
FIRT			

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
COUNT	00A5	165	TEMP USATO DA ROUTINE SERIALE
CNTDN			CONTO ALLA ROVESCIA DEL SINC. DELLA CASSETTA
BUFPT	00A6	166	PUNTATORE BUFFER DELLA CASSETTA
INBIT	00A7	167	MEMORIZ. BIT DI INPUT RCVR RS-232
SHCNL			CONTO BREVE CASSETTA
BITCI	00A8	168	CONTO BIT RCVR RS-232
RER			ERRORE DI LETTURA CASSETTA
RINONE	00A9	169	FLAG RCVR PER CONTROLLO BIT DI PARTENZA
REZ			ZERO LETTURA CASSETTA
RIDATA	00AA	170	BUFFER DEL BYTE RCVR RS-232
RDFLG			MODO LETTURA DI CASSETTA
RIPRTY	00AB	171	MEMORIZZ. DI PARITÀ RCVR RS-232
SHCNH			CNT BREVE CASSETTA
SAL	00AC	172	PUNTATORE: REGISTRA BUFFER/ SCROLL SCHERMO
SAH	00AD	173	
EAL	00AE	174	REGISTRA FINE INDIRIZZI/FINE PROGRAMMA
EAH	00AF	175	
CMP0	00B0	176	REGISTRA COSTANTI DEL TEMPO
TEMP	00B1	177	
TAPE1	00B2	178	INDIR. REGISTRAZ.DEL BUFFER
BITTS	00B4	180	CONTO BIT TRNS RS-232
SNSW1			
NXTBIT	00B5	181	RS-232 TRNS BIT SEGUENTE DA INVIARE
DIFF			
RODATA	00B6	182	RS-232 TRNS BYTE BUFFER
PRP			
FNLEN	00B7	183	LUNGHEZZA FILE CORRENTE N STR
LA	00B8	184	IND. CORRENTE FILE LOGICO
SA	00B9	185	2NDO IND. FILE CORRENTE
FA	00BA	186	IND. PRINCIPALE FILE CORRENTE
FNADR	00BB	187	NOME STR IND. FILE CORRENTE
ROPRTY	00BD	189	RS-232 TRNS BUFFER DI PARITÀ
OCHAR			
FSBLK	00BE	190	CONTO BLOCCO LETTURA DELLA CASSETTA
DRIVE	00BF	191	
MYCH			BUFFER PAROLA SERIALE
CAS1	00C0	192	CASSETTA MANUALE/INTERRUTTORE CONTROLLATO (AGGIORNATO DURANTE IRQ)
TRACK	00C1	193	
STAL			INDIRIZZO DI INIZIO I/O (BASSO)
SECTOR	00C2	194	
STAH			INDIR. DI INIZIO I/O (ALTO)

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
MEMUSS	**00C3**	**195**	**TEMP DI CARICAMENTO CASSETTA (2 BYTE)**
TMP2			
DATA	**00C5**	**197**	**REGISTRA LETTURA/SCRIVI DATI**
BA	**00C6**	**198**	**BANCO PER LE OPERAZIONI CORRENTI LOAD/SAVE/VERIFY**
FNBANK	**00C7**	**199**	**BANCO DOVE SI TROVA FN CORRENTE (A "FNADR")**
RIBUF	**00C8**	**200**	**RS-232 PUNTAT. BUFFER DI INPUT**
ROBUF	**00CA**	**202**	**RS-232 PUNTAT. BUFFER DI OUTPUT**
VARIABILI DELL'EDITOR GLOBALE DELLO SCHERMO			
KEYTAB	**00CC**	**204**	**PUNTATORE TABELLA KEYSCAN**
IMPARM	**00CE**	**206**	**PUNTATORE STRINGA UTILITY PRIMM**
NDX	**00D0**	**208**	**INDICE CODA TASTIERA**
KYNDX	**00D1**	**209**	**FLAG TASTO DELLA FUNZIONE IN SOSPESO**
KEYIDX	**00D2**	**210**	**INDICE NELLA STRINGA DEL TASTO DELLA FUNZIONE IN SOSPESO**
SHFLAG	**00D3**	**211**	**STATO TASTO SHIFT KEYSCAN**
SFDX	**00D4**	**212**	**INDICE KEYSCAN TASTO CORRENTE**
LSTX	**00D5**	**213**	**INDICE KEYSCAN ULTIMO TASTO**
CRSW	**00D6**	**214**	**FLAG INPUT <CR>**
MODE	**00D7**	**215**	**FLAG MODO 40/ 80 COLONNE**
GRAPHM	**00D8**	**216**	**FLAG MODO TESTO/GRAFICO**
CHAREN	**00D9**	**217**	**FLAG PRENDE CARATTERE RAM/ROM (BIT-2)**
LE SEGUENTI LOCAZIONI SONO CONDIVISE DA PARECCHIE ROUTINE EDITOR			
SEDSAL	**00DA**	**218**	**PUNTATORI DA MOVLIN**
BITMSK	**00DA**	**218**	**ROUTINE DI AVVOLGIMENTO TEMPORANEO PER TAB E LINE**
SAVER	**00DB**	**219**	**ALTRO POSTO TEMPORANEO PER SALVARE UN REG**
SEDEAL	**00DC**	**220**	
SEDT1	**00DE**	**222**	**SAVPOS**
SEDT2	**00DF**	**223**	
KEYSIZ	**00DA**	**218**	**VARIABILI DI TASTI PROGRAMMABILI**
KEYLEN	**00DB**	**219**	
KEYNUM	**00DC**	**220**	
KEYNXT	**00DD**	**221**	
KEYBNK	**00DE**	**222**	
KEYTMP	**00DF**	**223**	

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE

VARIABILI LOCALI DELL'EDITOR DELLO SCHERMO. QUESTE SONO SCAMBIATE IN $0A40 QUANDO CAMBIA IL MODO DELLO SCHERMO (40/80).

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
PNT	00E0	224	PUNTATORE ALLA LINEA CORRENTE (TESTO)
USER	00E2	226	PUNTATORE ALLA LINEA CORRENTE (ATTRIBUTO)
SCBOT	00E4	228	LIMITE BASSO DELLA FINESTRA
SCTOP	00E5	229	LIMITE ALTO DELLA FINESTRA
SCLF	00E6	230	MARGINE SINISTRO DELLA FINESTRA
SCTR	00E7	231	MARGINE DESTRO DELLA FINESTRA
LSXP	00E8	232	INIZIO COLONNA INPUT CORRENTE
LSTP	00E9	233	INIZIO LINEA INPUT CORRENTE
INDX	00EA	234	FINE LINEA INPUT CORRENTE
TBLX	00EB	235	LINEA CURSORE CORRENTE
PNTR	00EC	236	COLONNA CURSORE CORRENTE
LINES	00ED	237	NUM. MASSIMO DI LINEE DELLO SCHERMO
COLUMNS	00EE	238	NUM. MASSIMO DI COLONNE DELLO SCHERMO
DATAX	00EF	239	CARATTERE CORRENTE DA STAMPARE
LSTCHR	00F0	240	CAR. PRECED. STAMPATO (PER TESTO<ESC>)
COLOR	00F1	241	ATTRIBUTO CORR DA STAMPARE (COLORE DI DEFAULT FGND)
TCOLOR	00F2	242	ATTRIB SALVATO PER STAMPA ("INSERT" E "DELETE")
RVS	00F3	243	INVERSO MODO FLAG
QTSW	00F4	244	RICHESTA MODO FLAG
INSRT	00F5	245	INSERISCE MODO FLAG
INSFLG	00F6	246	AUTOINSERISCE MODO FLAG
LOCKS	00F7	247	DISABILITA <SHIFT><C=>, <CTRL>S
SCROLL	00F8	248	DISABILITA SCROLL DELLO SCHERMO ED IL LINKER DELLA LINEA
BEEPER	00F9	249	DISABILITA <CTRL>G
FREKZP	00FA	250	LIBERA LA PAG ZERO PER APPLICAZIONI SOFTWARE ($FA-$FE)
LOFBUF	00FF	255	

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
	VARS DELL'INTERFACCIA BASIC/DOS		
BAD	0100	256	REGISTRA GLI ERRORI LETTI
FBUFFER			AREA PER COSTRUIRE NOMI DI FILE (16 BYTE)
XCNT	0110	272	CONTATORE LOOP DOS
DOSF1L	0111	273	NOME DEL FILE DOS 1 LUN
DOSDS1	0112	274	DISK DRIVE DOS 1
DOSF2L	0113	275	NOME DEL FILE DOS 2 LUN
DOSDS2	0114	276	DISK DRIVE DOS 2
DOSF2A	0115	277	INDIR. DEL NOME DEL FILE DOS 2
DOSOFL	0117	279	INDIRIZZO DI PARTENZA DI BLOAD/ BSAVE
DOSOFH	0119	281	...E INDIR. FINALE
DOSLA	011B	283	INDIR LOGICO DOS
DOSFA	011C	284	INDIR FISICO DOS
DOSSA	011D	285	INDIR SEC. DOS
DOSRCL	011E	286	LUNGH. DEL RECORD DOS
DOSBNK	011F	287	
DOSDID	0120	288	DOS DISK ID
DIDCHK	0122	290	SPAZIO FLG DOS DISK ID USATO DALLA STAMPA
BNR	0123	291	PUNTATORE AL NUM INIZIALE
ENR	0124	292	PUNTATORE AL NUM FINALE
DOLR	0125	293	FLAG DOLLARO
FLAG	0126	294	FLAG VIRGOLA
SWE	0127	295	CONTATORE
USGN	0128	296	SEGNO DI ESPONENTE
UEXP	0129	297	PUNTATORE DELL'ESPONENTE
VN	012A	298	#CIFRE PRIMA DEL PUNTO DECIMALE
CHSN	012B	299	GIUSTIFICA FLAG
VF	012C	300	#POS PRIMA DEL PUNTO DECIMALE (CAMPO)
NF	012D	301	#POS DOPO IL PUNTO DECIMALE (CAMPO)
POSP	012E	302	+/- FLAG (CAMPO)
FESP	012F	303	FLAG ESPONENTE (CAMPO)
ETOF	0130	304	SCAMBIO
CFORM	0131	305	CONTATORE DEL CARATTERE (CAMPO)
SNO	0132	306	NO SEGNO
BLFD	0133	307	FLAG BLANK/STAR
BEGFD	0134	308	PUNTATORE A INIZIO CAMPO
LFOR	0135	309	LUNGH DEL FORMATO
ENDFD	0136	310	PUNTATORE A FINE CAMPO
SYSTK	0137	311	STACK DEL SISTEMA ($0137-$01FF)
BUF	0200	512	BUFFER DI INPUT: BASIC E MONITOR ($0200-$02A1)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
FETCH	02A2	674	LDA(-),Y DA QUALSIASI BANCO
FETVEC	02AA	682	
STASH	02AF	687	STA(-),Y IN QUALSIASI BANCO
STAVEC	02B9	697	
CMPARE	02BE	702	CMP(-),Y IN QUALSIASI BANCO
CMPVEC	02C8	712	
JSRFAR	02CD	716	JSR XXXX IN QUALSIASI BANCO E RITORNO
JMPFAR	02E3	739	JMP XXXX IN QUALSIASI BANCO
		VETTORI	
ESC-FN-VEC	02FC	764	VETTORE PER ROUTINE DI FUNZIONI IN PIÙ
BNKVEC	02FE	766	VETTORE PER UTENTI DI FUNZIONE CART.
IERROR	0300	768	VETTORE PER STAMPA ERRORE BASIC (ERR IN .X)
IMAIN	0302	770	VETTORE AL PRINCIPALE (LOOP DIRETTO DEL SISTEMA)
ICRNCH	0304	772	VETTORE DA SCHIACCIARE (ROUTINE DI SIMBOLIZZAZ.)
IQPLOP	0306	774	VETTORE PER LISTARE IL TESTO BASIC (CHAR LIST)
IGONE	0308	776	VETTORE A GONE (DISPACCIO IN CAR BASIC)
IEVAL	030A	778	VETTORE PER LA VALUTAZIONE DI SEGNI BASIC
IESCLK	030C	780	VETTORE PER I SEGNI ESCAPE CRUNCH,
IESCPR	030E	782	...LIST,
IESCEX	0310	784	...E ESEGUI.
IIRQ CINV	0314	788	VETTORE IRQ RAM
IBRK CBINV	0316	790	VETTORE RAM BRK INSTR
INMI	0318	792	VETTORE NMI
IOPEN	031A	794	VETTORE DELLA ROUTINE KERNAL DI APERTURA
ICLOSE	031C	796	VETTORE DELLA ROUTINE KERNAL DI CHIUSURA
ICHKIN	031E	798	VETTORE DELLA ROUTINE KERNAL CHKIN
ICKOUT	0320	800	VETTORE DELLA ROUTINE KERNAL CHKOUT
ICLRCH	0322	802	VETTORE DELLA ROUTINE KERNAL CLRCHN
IBASIN	0324	804	VETTORE DELLA ROUTINE KERNAL CHRIN

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
IBSOUT	0326	806	VETTORE DELLA ROUTINE KERNAL CHROUT
ISTOP	0328	808	VETTORE DELLA ROUTINE KERNAL STOP
IGETIN	032A	810	VETTORE DELLA ROUTINE KERNAL GETIN
ICLALL	032C	812	VETTORE DELLA ROUTINE KERNAL CLALL
EXMON	032E	814	VETTORE DEL COMANDO DEL MONITOR
ILOAD	0330	816	VETTORE DELLA ROUTINE KERNAL DI CARICAMENTO
ISAVE	0332	818	VETTORE DELLA ROUTINE KERNAL DI SALVATAGGIO
			VETTORI INDIRETTI DELL'EDITOR
CTLVEC	0334	820	EDITOR:STAMPA L'INDIRETTO "CONTRL"
SHFVEC	0336	822	EDITOR:STAMPA L'INDIRETTO "SHIFTD"
ESCVEC	0338	824	EDITOR:STAMPA L'INDIRETTO "ESCAPE"
KEYVEC	033A	826	EDITOR:KEYSCAN DELL'INDIRETTO LOGICO
KEYCHK	033C	828	EDITOR:MEMORIZ L'INDIRETTO DEL TASTO
DECODE	033E	830	VETTORI DELLE TABELLE DI DECODIFI-CAZ. MATRICI DELLA TASTIERA
KEYD	034A	842	BUFFER IRQ TASTIERA (10 BYTE)
TABMAP	0354	852	BITMAP DEGLI STOP TAB (10 BYTE,$0354-D)
BITABL	035E	862	BITMAP DELLE LINEE AVVOLTE TAB-MAP BITABL SONO SPOSTATE IN $0A60 QUANDO IL MODO SCHERMO 40/80 È CAMBIATO.
LAT	0362	866	NUMERI FILE LOGICI
FAT	036C	876	NUMERI DELLE UNITÀ PRIMARIE
SAT	0376	886	INDIR SECOND
CHRGET	0380	896	
CHRGOT	0386	902	
QNUM	0390	912	
			AREA DELLE SUBROUTINE DI CARICAMENTO DEGLI INDIRETTI
INDSUB-RAM0	039F	927	SUB PRESA NELLA ROM CONDIVISA
INDSUB-RAM1	03AB	939	SUB PRESA NELLA ROM CONDIVISA
INDIN1-RAM1	03B7	950	PRENDE L'INDIRETTO INDICE 1

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
INDIN2	03C0	959	PRENDE L'INDIRETTO INDICE 2
INDTXT	03C9	968	TXTPTR
ZERO	03D2	977	COSTANTE NUMER PER IL BASIC
CURRENT-BANK	03D5	979	CONTESTO PER SYS,POKE,PEEK DAL BANCO CMMD
TMPDES	03D6	980	TEMP PER ISTR
FIN-BANK	03DA	984	PUNTATORE DEL BANCO PER LA CONVESIONE STRINGA/NUMERO RTN
SAVIZ	03DB	985	LOCAZIONI DI OPERAZ TEMP PER SSHAPE
BITS	03DF	989	CIFRA OVERFLOW FAC#1
SPRTMP-1	03E0	990	TEMP PER SPRSAV
SPRTMP-2	03E1	991	
FG-BG	03E2	992	NYBBLE IMPACCATI DEL COL DI PRIMO PIANO/SFONDO
FG-MC1	03E3	993	NYBBLE IMPACCATI DEL COL DI PRIMO PIANO/MULTICOL 1

DICHIARAZIONI DI PAGINA QUATTRO E OLTRE

			(INIZIO DELLA RAM A BANCHI)
VICSCN	0400	1024	MATRICE VIDEO #1:VIC TESTO SCHERMO A 40 COLONNE $0400-$07FF
	0800	2048	STACK RUN-TIME BASIC (512 BYTE) $0800-$09FF

VARIABILI KERNAL ASSOLUTE

SYSTEM-VECTOR	0A00	2560	VETTORE DI NUOVO INIZIO DI SISTEMA (BASIC CALDO)
DEJAVU	0A02	2562	BYTE DI STATO DI INIZ KERNAL CALDO/FREDDO
PALNTS	0A03	2563	FLAG DI SISTEMA PAL/NTSC
INIT-STATUS	0A04	2564	RESET FLAG CONTRO STATO NMI PER INIZ RTNS
MEMSTR	0A05	2565	PTR IN FONDO MEM DISPONIB NEL BANCO DEL SISTEMA
MEMSIZ	0A07	2567	PTR IN CIMA MEM DISPONIB NEL BANCO DEL SISTEMA
IRQTMP	0A09	2569	GESTORE NASTRO PRESERVA QUI INDIRETTO IRQ
CASTON	0A0B	2571	SENSO TOD DURANTE LE OPERAZ NASTRO
KIKA26	0A0C	2572	NASTRO LEGGE TEMPORANEAM
STUPID	0A0D	2573	NASTRO LEGGE INDICATORE D1IRQ
TIMOUT	0A0E	2574	FLAG TIMEOUT SERIALE VELOC
ENABL	0A0F	2575	ABILITA RS-232

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
M51CTR	0A10	2576	REGISTRO DI CONTROLLO RS-232
M51CDR	0A11	2577	REGISTRO DI COMANDO RS- 232
M51AJB	0A12	2578	BAUD RATE DELL'UTENTE RS-232
RSSTAT	0A14	2580	REGISTRO DI STATO RS-232
BITNUM	0A15	2581	NUMERO DI BIT DA INIVIARE RS-232
BAUDOF	0A16	2582	BAUD RATE BIT A TEMPO PIENO RS-232 (CREATO DA OPEN)
RIDBE	0A18	2584	INDICE DEL BUFFER DI INPUT RS-232 ALLA FINE
RIDBS	0A19	2585	INDICE DEL BUFFER DI INPUT RS-232 ALL'INIZIO
RODBS	0A1A	2586	INDICE DEL BUFFER DI OUTPUT RS-232 ALL'INIZIO
RODBE	0A1B	2587	INDICE DEL BUFFER DI OUTPUT RS-232 ALLA FINE
SERIAL	0A1C	2588	FLAG SERIALE VEL INTERNO/ESTERNO
TIMER	0A1D	2589	DECREMENTA REGISTRO JIFFIE

DICHIARAZIONI DELL'EDITOR DELLO SCHERMO ASSOLUTE GLOBALI

XMAX	0A20	2592	MISURA MAX DELLA CODA TASTIERA
PAUSE	0A21	2593	FLAG <CTRL>
RPTFLG	0A22	2594	ABILITA LE RIPETIZIONI DEL TASTO
KOUNT	0A23	2595	RITARDO TRA RIPETIZIONI DEL TASTO
DELAY	0A24	2596	RITARDO PRIMA CHE UN TASTO SIA RIPETUTO
LSTSHF	0A25	2597	RITARDO TRA <C=> <SHIFT>
BLNON	0A26	2598	MODO CURSORE VIC (LAMPEGGIO SOLIDO)
BLNSW	0A27	2599	DISABILITA CURSORE VIC
BLNCT	0A28	2600	CONTATORE LAMPEGGIO CURSORE VIC
GDBLN	0A29	2601	CAR CURSORE VIC PRIMA DI LAMPEGGIO
GDCOL	0A2A	2602	COLORE DEL CURSORE VIC PRIMA DEL LAMPEGGIO
CURMOD	0A2B	2603	MODO CURSORE VDC (SE ABILTATO)
VM1	0A2C	2604	PUNTATORE BASE TESTO SCHERMO/ CAR VIC
VM2	0A2D	2605	PUNTATORE BASE BIT MAP VIC
VM3	0A2E	2606	BASE TESTO SCHERMO VDC
VM4	0A2F	2607	ATTRIBUTO BASE VDC

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
LINTMP	0A30	2608	PUNTATORE TEMPORANEO A ULTIMA LINEA PER LOOP4
SAV80A	0A31	2609	ROUTINE TEMPORANEE PER 80 COL
SAV80B	0A32	2610	ROUTINE TEMPORANEE PER 80 COL
CURCOL	0A33	2611	COLORE CURSORE VDC PRIMA DEL LAMPEGGIO
SPLIT	0A34	2612	VALORE DEL RASTER DI DIVISIONE SCHERMO VIC
FNADR	0A35	2613	SALVA .X DURANTE LE OPERAZIONI DI BANCO
PALCNT	0A36	2614	CONTATORE PER SISTEMI PAL (REGOLAZ JIFFIE)
SPEED	0A37	2615	SALVA LA VELOCITÀ DEL SISTEMA DURANTE LE OPS NASTRO E SERIALI
SPRITES	0A38	2616	SALVA LE ABILITAZ DI SPRITE DURANTE LE OPS NASTRO E SERIALI
BLANKING	0A39	2617	SALVA LO STATO DI LAMPEG DURANTE LE OPS DI NASTRO
HOLD-OFF	0A3A	2618	FLAG POSTO DA UTENTE PER PIENO CNTRL DEL VIC
LDTB1-SA	0A3B	2619	BYTE ALTO: SA DI SCHER VIC (USA W/ VM1 PER MUOVERE SCHERMO
CLR-EA-LO	0A3C	2620	RIEMPIE BLOCCO 8563
CLR-EA-HI	0A3D	2621	RIEMPIE BLOCCO 8563
	0A40	2624	AREA DI SWAP RISERVATA $0A40-$0A7F PER VARIAB SCHER QUANDO MODO (40/80) CAMBIA MONITOR
XCNT	0A80	2688	COMPARA I BUFFER (32 BYTE)
HULP	0AA0	2720	
FORMAT	0AAA	2730	
LENGHT	0AAB	2731	ASM/DIS
MSAL	0AAC	2732	PER ASSEMBLER
SXREG	0AAF	2735	1 BYTE TEMP USATO OVUNQ
SYREG	0AB0	2736	1 BYTE TEMP USATO OVUNQ
WRAP	0AB1	2737	1 BYTE TEMP PER ASSEMBLER
XSAVE	0AB2	2738	SALVA .X QUI DUR CHIAMATE SUBROUTINE INDIRETTI
DIRECTION	0AB3	2739	INDIC DI DIREZ PER “TRANSFER“
COUNT	0AB4	2740	CONVERSIONE NUM PARSE
NUMBER	0AB5	2741	CONVERSIONE NUM PARSE
SHIFT	0AB6	2742	CONVERSONE NUM PARSE
TEMPS	0AB7	2743	

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
			TABELLE CARD DEI TASTI DI FUNZIONE DELLA ROM
CURBNK	0AC0	2752	TASTO DI FUNZIONE CORRENTE DEL BANCO ROM È INTERROGATO
PAT	0AC1	2753	TABELLA DI INDIR FISICO (IDS DI CARDS REGISTRATE)
DK-FLAG	0AC5	2757	RISERVATO PER EDITOR DI SCHERMI ESTRANEI
	0AC6	2758	$0AC6-$0AFF RISERVATO PER I SISTEMI
TBUFFR	0B00	2810	BUFFER DELLE CASSETTE (192 BYTE) $0B00-$0BC0 QUESTA PAG USATA ANCHE COME BUFFER PER l'AUTOBOOT DEL DISCHETTO
RS232I	0C00	3072	RS-232 BUFFER DI INPUT
RS232O	0D00	3328	RS-232 BUFFER DI OUTPUT
	0E00	3584	AREA DEFINIZ DELLO SPRITE (DEVE ESSERE SOTTO $1000) $0E00-$0FFF, 512 BYTE
PKYBUF	1000	4096	TABELLA DI LUNGH DEI TASTI DI FUNZ PROGRAMMABILI PER 10 TASTI (F1-F8,<SHIFT RUN>,HELP)
PKYDEF	100A	4106	STRINGHE DEI TASTI DELLE FUNZIONI PROGRAMMABILI
			AREA DOS/VSP
DOSSTR	1100	4352	DOS OUTPUT STR. BUF 48 BYTE PER COSTRUIRE UNA STRINGA DOS
VWORK	1131	4401	VARS GRAFICHE
XYPOS	1131	4401	
XPOS	1131	4401	POSIZ X CORRENTE
YPOS	1133	4403	POSIZ Y CORRENTE
XDEST	1135	4405	DESTINAZI COORD X
YDEST	1137	4407	DESTINAZI COORD Y
XYABS	1139	4409	VARIABILI DISEGNO LINEA
XABS	1139	4409	
YABS	113B	4411	
XYSGN	113D	4413	
XSGN	113D	4413	
YSGN	113F	4415	
FCT	1141	4417	
ERRVAL	1145	4421	
LESSER	1147	4423	
GREATR	1148	4424	
			VARIABILI ROUTINE DI ANGOLO
ANGSGN	1149	4425	SEGNO DI ANGOLO
SINVAL	114A	4426	SENO DEL VALORE DELL'ANGOLO

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
COSVAL	114C	4428	COSENO DEL VALORE DELL'ANGOLO
ANGCNT	114E	4430	TEMP PER LE ROUTINE DI DISTANZA DELL'ANGOLO

VARIABILI GRAFICHE BASIC.
I SEGUENTI 24 BYTE SONO DEFINITI IN MODO MULTIPLO.

			VARIABILI CHE DISEGNANO UN CERCHIO
XCIRCL	1150	4432	CENTRO DEL CERCHIO, COORD X
YCIRCL	1152	4434	CENTRO DEL CERCHIO, COORD Y
XRADUS	1154	4436	RAGGIO X
YRADUS	1156	4438	RAGGIO Y
ROTANG	1158	4440	ANGOLO DI ROTAZIONE
ANGBEG	115C	4444	INIZIO ARC ANGOLO
ANGEND	115E	4446	FINE ARC ANGOLO
XRCOS	1160	4448	RAGGIO X* COS(ANG DI ROTAZIONE)
YRSIN	1162	4450	RAGGIO Y* SEN(ANG DI ROTAZIONE)
XRSIN	1164	4452	RAGGIO X* SEN(ANG DI ROTAZIONE)
YRCOS	1166	4454	RAGGIO Y* COS(ANG DI ROTAZIONE)

PARAMETRI BASIC DI USO GENERALE

XCENTR	1150	4432	
YCENTR	1152	4434	
XDIST1	1154	4436	
YDIST1	1156	4438	
XDIST2	1158	4440	
YDIST2	115A	4442	
DISEEND	115C	4444	CONSERVA POS
COLCNT	115E	4446	CONTATORE COL DEL CAR
ROWCNT	115F	4447	
STRCNT	1160	4448	

			VARIABILI DEL DISEGNO DELLA CASELLA
XCORD1	1150	4432	PUNTO 1 COORD X
YCORD1	1152	4434	PUNTO 1 COORD Y
BOXANG	1154	4436	ANGOLO DI ROTAZIONE
XCOUNT	1156	4438	
YCOUNT	1158	4440	
BXLENG	115A	4442	LUNG DI UN LATO
XCORD2	115C	4444	
YCORD	115E	4446	

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
			VARIABILI DELLA FORMA E DI MOVIMENTO DELLA FORMA
KEYLEN	1151	4433	
KEYNXT	1152	4434	
STRSZ	1153	4435	LUNG STRINGA
GETTYP	1154	4436	RIPONE IL MODO SHAPE
STRPTR	1155	4437	CONTATORE POS DELLA STRINGA
OLDBYT	1156	4438	BYTE VECCHIO BIT MAP
NEWBIT	1157	4439	NUOVA STRINGA O BYTE DEL BIT MAP
	1158	4440	CONSERVATORE DEL POSTO
XSIZE	1159	4441	LUNG COLONNA SHAPE
YSIZE	115B	4443	LUNG RIGA SHAPE
XSAVE	115D	4445	TEMP PER LA LUNG COLONNA
STRADR	115F	4447	SALVA IL DESCRITTORE DELLA STRINGA SHAPE
BITIDX	1161	4449	INDICE BIT NEL BYTE
			VARIABILI BASIC GRAFICHE
CHRPAG	1168	4456	BYTE ALTO: INDIR CHARROM PER CMD "CHAR"
BITCNT	1169	4457	TEMP PER GSHAPE
SCALEM	116A	4458	FLAG MODO SCALE
WIDTH	116B	4459	FLAG A DOPPIA LARG
FILFLG	116C	4460	FLAG RIEMPI RETTANGOLO
BITMSK	116D	4461	TEMP PER LA MASCHERA DEL BIT
NUMCNT	116E	4462	
TRCFLG	116F	4463	MODO TRACCIA FLAG
RENUM-TMP-1	1170	4464	UN TEMP PER RINUMERARE
RENUM-TMP-2	1172	4466	UN TEMP PER RINUMERARE
T3	1174	4468	
T4	1175	4469	
VTEMP3	1177	4471	MEMORIZ DEL TEMP GRAFICO
VTEMP4	1178	4472	
VTEMP5	1179	4473	
ADRAY1	117A	4474	PTR ALLA ROUTINE: CONVERTE FLOAT → INTEGER
ADRAY2	117C	4476	PTR ALLA ROUTINE: CONVERTE INTEGER → FLOAT
SPRITE-DATA	117E	4478	TABELLE DI VELOC/DIREZ DI SPRITE ($117E-D5)
VIC-SAVE	11D6	4566	COPIA DEL REG VIC USATA PER AGGIORNARE CHIP DURANTE RINTRACCIAM (21 BYTE, $11D6-EA)
UPPER-LOWER	11EB	4587	PUNTATORE ALL'INSIEME DEL CAR SUPERIORE/ INFERIORE PER IL CAR
UPPER-GRAPHIC	11EC	4588	PTR ALL'INS DI CARATTERI SUPERIORI/ GRAFICI

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
DOSSA	11ED	4589	TEMP MEMORIZ PER FILE SA DURANTE CMD RECORD
OLDLIN	1200	4608	NUMERO DELLA LINEA PRECED BASIC
OLDTXT	1202	4610	PUNTATORE: FRASE BASIC CONTINUA

DICHIARAZIONI PER L'USO DELLA STAMPANTE

PUCHRS	1204	4612	
PUFILL	1204	4612	STAMPA USANDO IL SIMBOLO RIEMPIRE
PUCOMA	1205	4613	STAMPA USANDO IL SIMBOLO VIRGOLA
PUDOT	1206	4614	STAMPA USANDO IL SIMBOLO D.P.
PUMONY	1207	4615	STAMPA USANDO IL SIMBOLO MONETARIO
ERRNUM	1208	4616	USATO DALLA ROUTINE INTRAPPO-LAM. DI ERRORE- NUM ULTIMO ERRO-RE
ERRLIN	1209	4617	LINEA# DI ULTIMO ERR. $FFFF SE NON ERRORI
TRAPNO	120B	4619	LINEA VA SU ERRORE.$FFXX SE NESSU-NO È POSTO
TMPTRP	120D	4621	TIENE TRAP# TEMPORANEAM
ERRTXT	120E	4622	
TEXT-TOP	1210	4624	CIMA DEL PUNTATORE DEL TESTO
MAX-MEM-0	1212	4626	INDIR DISP PIÙ ALTO IN BASIC NELLA RAM 0
TMPTXT	1214	4628	USATO DA DO-LOOP PUÒ ESSERE MULT ASSEGNATO
TMPLIN	1216	4630	
SRPOK	1218	4632	
RNDX	121B	4635	
CIRCLE-SEGMENT	1220	4640	GRADI PER SEGMENTO DI CERCHIO
DEJAVU	1221	4641	RESET STATO "FREDDO" O "CALDO"

MEMORIZZAZIONE BASIC PER I VETTORI MUSICALI

TEMPO-RATE	1222	4642
VOICES	1223	4643
NTIME	1229	4649
OCTAVE	122B	4651
SHARP	122C	4652
PITCH	122D	4653
VOICE	122F	4655
WAVE0	1230	4656
DNOTE	1233	4659
FLTSAV	1234	4660
FLTFLG	1238	4664
NIBBLE	1239	4665
TONNUM	123A	4666

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
TONVAL	123B	4667	
PARCNT	123E	4668	
ATKTAB	123F	4669	
SUSTAB	1249	4681	
WAVTAB	1253	4691	
PULSLW	125D	4701	
PULSHI	1267	4711	
FILTERS	1271	4721	
		VETTORI DI INTERRUZIONE	
INT-TRIP-FLAG	1276	4726	
INT-ADR-LO	1279	4729	
INT-ADR-HI	127C	4732	
INTVAL	127F	4735	
COLTYP	1280	4736	
		VARS DEI COMANDI DEL SUONO IN BASIC	
SOUND-VOICE	1281	4737	
SOUND-TIME-LO	1282	4738	
SOUND-TIME-HI	1285	4741	
SOUND-MAX-LO	1288	4744	
SOUND-MAX-HI	128B	4747	
SOUND-MIN-LO	128E	4750	
SOUND-MIN-HI	1291	4753	
SOUND-DIRECTION	1294	4756	
SOUND-STEP-LO	1297	4759	
SOUND-STEP-HI	129A	4762	
SOUND-FREQ-LO	129D	4765	
SOUND-FREQ-HI	12A0	4768	
TEMP-TIME-LO	12A3	4771	
TEMP-TIME-HI	12A4	4772	
TEMP-MAX-LO	12A5	4773	
TEMP-MAX-HI	12A6	4774	
TEMP-MIN-LO	12A7	4775	
TEMP-MIN-HI	12A8	4776	
TEMP-DIRECTION	12A9	4777	
TEMP-STEP-LO	12AA	4778	
TEMP-STEP-HI	12AB	4779	
TEMP-FREQ-LO	12AC	4780	
TEMP-FREQ-	12AD	4781	
HITEMP-PULSE-LO	12AE	4782	
TEMP-PULSE-HI	12AF	4783	
TEMP-WAVEFORM	12B0	4784	

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
POT-TEMP-1	12B1	4785	TEMP PER LA FUNZIONE "POT"
POT-TEMP-2	12B2	4786	
WINDOW-TEMP	12B3	4787	
SAVRAM	12B7	4791	USATO DA SPRDEF E SAVSPR
DEFMOD	12FA	4858	USATO DA SPRDEF E SAVSPR
LINCTN	12FB	4859	USATO DA SPRDEF E SAVSPR
SPRITE-NUMBER	12FC	4860	USATO DA SPRDEF E SAVSPR
IRQ-WRAP-FLAG	12FD	4861	USATO DA IRQ BASIC PER BLOCCARE TUTTO TRANNE UNA CHIAMATA IRQ
	1300	4864	AREA DI APPLICAZ DEL PROGRAMMA $1300-$1BFF
RAMBOT	1C00	7168	INIZIO TESTO BASIC $1C00- $EFFF (KERNAL PONE QUI MEMBOT) O
	1C00	7168	MATRICE VIDEO #2 (1KB DI COL PER BITMAP, SE ALLOCATI) $1C00-$1FFF
	2000	8192	BITMAP VIC 8KB, SE ALLOCATI) $2000-$3FFF
			INIZIO DELLA ROM SULLA RAM
	4000	16384	ROM BASSA BASIC C128 INIZIO TESTO BASIC SE BIT MAP È ALLOC (RAM)$4000- $EFFF
	8000	32768	ROM ALTA BASIC C128 (O FUNZIONE ROM)$8000- $BFFF

TABELLA DI SALTO IN BASIC

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
			ENTRATE BASIC
JMP HARD-RESET	4000	16384	ENTRATA FREDDA
JMP SOFT-RESET	4003	16387	ENTRATA CALDA
JMP BASIC-IRQ	4006	16390	ENTRATA IRQ
			CONVERSIONI DEL FORMATO
JMP AYINT	AF00	44800	CONV. F.P. IN INTERO
JMP GIVAYF	AF03	44803	CONV INTERO IN F.P.

Tabella di salto in basic (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
JMP FOUT	AF06	44806	CONV F.P. IN STRINGA ASCII
JMP VAL-1	AF09	44809	CONV STRINGA ASCII IN F.P.
JMP GETADR	AF0C	44812	CONV F.P. IN UN INDIRIZZO
JMP FLOATC	AF0F	44815	CONV INDIRIZZO IN F.P.
		FUNZIONI MATEMATICHE	
JMP FSUB	AF12	44818	MEM-FACC
JMP FSUBT	AF15	44821	ARG-FACC
JMP FADD	AF18	44824	MEM+FACC
JMP FADDT	AF1B	44827	ARG+FACC
JMP FMULT	AF1E	44830	MEM*FACC
JMP FMULTT	AF21	44833	ARG*FACC
JMP FDIV	AF24	44836	MEM/FACC
JMP FDIVT	AF27	44839	ARG/FACC
JMP LOG	AF2A	44842	CALCOLA LOG NATURALE DI FACC
JMP INT	AF2D	44845	ESEGUE INT BASIC DI FACC
JMP SQR	AF30	44848	CALCOLA RADICE QUADRATA DI FACC
JMP NEGOP	AF33	44851	NEGA FACC
JMP FPWR	AF36	44854	ELEVA ARG ALLA POTENZA IN MEM
JMP FPWRT	AF39	44857	ELEVA ARG ALLA POTENZA FACC
JMP EXP	AF3C	44860	CALCOLA L'ESP DI FACC
JMP COS	AF3E	44863	CALCOLA IL COS DI FACC
JMP SIN	AF42	44866	CALCOLA IL SEN DI FACC
JMP TAN	AF45	44869	CALCOLA LA TAN DI FACC
JMP ATN	AF48	44872	CALCOLA ATN DI FACC
JMP ROUND	AF4B	44875	ARROTONDA FACC
JMP ABS	AF4E	44878	VAL ASSOLLUTO DI FACC
JMP SIGN	AF51	44881	TEST DEL SEGNO DI FACC
JMP FCOMP	AF54	44884	COMPARA FACC CON MEM
JMP RND 0	AF57	44887	GENERA UN NUMERO A CASO DI F.P.
		MOVIMENTO	
JMP CONUPK	AF5A	44890	MUOVE MEM RAM VERSO ARG
JMP ROMUPK	AF5D	44893	MUOVE MEM ROM VERSO ARG
JMP MOVFRM	AF60	44896	MUOVE MEM RAM VERSO FACC
JMP MOVFM	AF63	44899	MUOVE MEM ROM VERSO FACC
JMP MOVMF	AF66	44902	MUOVE FACC VERSO MEM
JMP MOVFA	AF69	44905	MUOVE ARG VERSO FACC
JMP MOVAF	AF6C	44908	MUOVE FACC VERSO ARG

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
	ALTRE ROUTINE IN BASIC		
JMP OPTAB	AF6F	44911	
JMP DRAWLN	AF72	44914	
JMP GPLOT	AF75	44917	
JMP CIRSUB	AF78	44920	
JMP RUN	AF7B	44923	
JMP RUNC	AF7E	44926	
JMP CLEAR	AF81	44929	
JMP NEW	AF84	44932	
JMP LNKPRG	AF87	44935	
JMP CRUNCH	AF8A	44938	
JMP FNDLIN	AF8D	44941	
JMP NEWSTT	AF90	44944	
JMP EVAL	AF93	44947	
JMP FRMEVL	AF96	44950	
JMP RUN-A-PROGRAM	AF99	44953	
JMP SETEXC	AF9C	44956	
JMP LINGET	AF9F	44959	
JMP GARBA2	AFA2	44962	
JMP EXECUTE -A-LINE	AFA5	44965	
	ENTRATE MONITOR		
JMP CALL	B000	45056	CHIAMATA DI ENTRATA MONITOR
JMP BREAK	B003	45059	ROTTURA DI ENTRATA MONITOR
JMP MONCMD	B006	45062	COMANDO PARSER DI ENTRATA MONITOR
	C000	49152	ROM KERNAL (O FUNZIONE) $C000-$FFFF

TABELLA DI SALTO DELL'EDITOR

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
JMP CINT	C000	49152	INIZIALIZZA EDITOR E SCHERMO
JMP DISPLY	C003	49155	VISUAL CAR IN .A, COL IN .X
JMP LP2	C006	49158	PONE TASTO DAL BUFFER IRQ IN .A
JMP LOOP5	C009	49161	PONE UN CAR DALLA LINEA SCHERMO IN .A
JMP PRINT	C00C	49164	STAMPA CAR IN .A
JMP SCRORG	C00F	49167	PONE #RIGHE SCHERMO. COL IN X & Y

Tabella di salto dell'Editor (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
JMP SCNKEY	C012	49170	SUBROUTINE DI ANALISI TASTIERA
JMP REPEAT	C015	49173	GESTIONE DEI TASTI REPEAT E MEMORIZ DECODIFICATA
JMP PLOT	C018	49176	LEGGE O PONE POS CURSORE IN .X,.Y
JMP CURSOR	C01B	49179	SUB MUOVE CURSORE 8563
JMP ESCAPE	C01E	49182	ESEGUE FUNZ ESC USANDO CHR IN .A
JMP KEYSET	C021	49185	RIDEFINISCE UN TASTO A FUNZIONE PROGRAMMABILE
JMP IRQ	C024	49188	ENTRATA IRQ
JMP INIT80	C027	49191	INIZ INSIEME CAR AD 80 COLONNE
JMP SWAPPER	C02A	49194	CAMBIA GLI EDITOR LOC (CAMBIO 40/80)
JMP WINDOW	C02D	49197	PONE L'ALTO SINISTRA O IL BASSO A DESTRA DELLA FINESTRA
	D000	53248	ROM CAR VIC ($D000-$DFFF)

REGISTRI DEL CHIP VIC

VICREG0	D000	53248	SPRITE 0, LOCAZIONE X
VICREG1	D001	53249	SPRITE 0, LOC Y
VICREG2	D002	53250	SPRITE 1, LOC X
VICREG3	D003	53251	SPRITE 1, LOC Y
VICREG4	D004	53252	SPRITE 2, LOC X
VICREG5	D005	53253	SPRITE 2, LOC Y
VICREG6	D006	53254	SPRITE 3, LOC X
VICREG7	D007	53255	SPRITE 3, LOC Y
VICREG8	D008	53256	SPRITE 4, LOC X
VICREG9	D009	53257	SPRITE 4, LOC Y
VICREG10	D00A	53258	SPRITE 5, LOC X
VICREG11	D00B	53259	SPRITE 5, LOC Y
VICREG12	D00C	53260	SPRITE 6, LOC X
VICREG13	D00D	53261	SPRITE 6, LOC Y
VICREG14	D00E	53262	SPRITE 7, LOC X
VICREG15	D00F	53263	SPRITE 7, LOC Y
VICREG16	D010	53264	MSBIT DI LOC X PER GLI SPRITE 0-7

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE

REGISTRI DEL CHIP VIC

VICREG17	D011	53265	REG 1 VIC DI CONTROLLO

7	BIT COMPARA RASTER
6	MODO TESTO COL ESTESO (1=ON)
5	MODO BIT MAP (1=ABILITA)
4	SCR BLANK AL COL DEI BORDI (0= BLANK)
3	SELEZ VIDEO TESTO 24/25 RIGHE(1=25)
2-0	SCROLL DOLCE FINO A POS Y

VICREG18	D012	53266	SCRIVE/LEGGE VAL RASTER PER COMPARARE IRQ
VICREG19	D013	53267	LATCH POS X PENNA OTTICA
VICREG20	DO14	53268	LATCH POS Y PENNA OTTICA
VICREG21	DO15	53269	ABILITA VIDEO SPRITE 0-7 (1 =ABILITA)
VICREG22	DO16	53270	REG 2 VIC DI CONTROLLO BIT

7-6	NON USATI
5	RESET
4	MODO MULTI COL (1=ABIL)
3	SELEZ VIDEO 38/40 COL (1=40COL)
2-0	SCROLL DOLCE A POS X

VICREG23	D017	53271	ESPANDE Y SPRITE 0-7
VICREG24	D018	53272	REG VIC DI CONTROL MEM BIT ($D018)

7-4	INDIR BASE MATR VIDEO
3-0	INDIR BASE CAR DOT-DATA

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
			REGISTRI DEL CHIP VIC
VICREG25	D019	53273	REG VIC DI INTERRUZ FLAG (1=IRQ) 7 — PONE QUALS CONDIZ IRQ VIC ABILIT 6-4 — NON USATO 3 — FLAG IRQ DELLA PENNA OTTICA TRIGGERATO 2 — FLAG IRQ DI COLLISIONE SPRITE CON SPRITE 1 — FLAG IRQ DI COLLISIONE SPRITE CON SFONDO 0 — FLAG IRQ DI COMPARAZ RASTER
VICREG26	D01A	53274	BIT ABILITAZ IRQ (1= ABILITATI) 7-4 — NON USATI 3 — PENNA OTT 2 — SPRITE VERSO SPRITE 1 — SPRITE VERSO SFONDO 0 — IRQ
VICREG27	D01B	53275	PRIORITÀ DI SFONDO SPRITE 0-7 (1=SPRITE)
VICREG28	D01C	53276	MODO MULTI COLORE SPRITE 0-7 (1=MULTICOL)
VICREG29	D01D	53277	ESPANDE X SPRITE 0-7
VICREG30	D01E	53278	LATCH DI COLLISIONE SPRITE CONTRO SPRITE
VICREG31	D01F	53279	LATCH DI COLLISIONE SPRITE CONTRO SFONDO
VICREG32	D020	53280	COLORE DEL BORDO
VICREG33	D021	53281	COL DI FONDO 0
VICREG34	D022	53282	COL FONDO 1
VICREG35	D023	53283	COL FONDO 2
VICREG36	D024	53284	COL FONDO 3
VICREG37	D025	53285	REG 0 MULTICOL DELLO SPRITE
VICREG38	D026	53286	REG 1 MULTICOL DELLO SPRITE
VICREG39	D027	53287	COL SPRITE 0
VICREG40	D028	53288	COL SPRITE 1
VICREG41	D029	53289	COL SPRITE 2
VICREG42	D02A	53290	COL SPRITE 3

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
			REGISTRI DEL CHIP VIC
VICREG43	D02B	53291	COL SPRITE 4
VICREG44	D02C	53292	COL SPRITE 5
VICREG45	D02D	53293	COL SPRITE 6
VICREG46	D02E	53294	COL SPRITE 7
VICREG47	D02F	53295	LINEE DELLA TASTIERA BIT 7-3 NON USATI 2-0 K2,K1 E K0
VICREG48	D030	53296	VELOCITÀ DEL CLOCK BIT 7-2 NON USATI 1 TEST 0 2 MHZ
			REGISTRI SID
SIDREG0	D400	54272	VOCE 1 FREQ BASSA
SIDREG1	D401	54273	VOCE 1 FREQ ALTA
SIDREG2	D402	54274	VOCE 1 LARGH D'IMPULSO BASSA
SIDREG3	D403	54275	VOCE 1 LARGH D'IMPULSO ALTA (0-15)
SIDREG4	D404	54276	VOCE 1 REG DI CONTROLLO 7 RUMORE (1= RUMORE) 6 IMPULSO (1 =IMPULSO) 5 SEGA (1= DENTE DI SEGA) 4 TRI (1= TRIANGOLO) 3 TEST (1= DISABILITA OSCILLATORE) 2 SUONO (1= MODUL SUONO OSC 1 CON OUTPUT OSC 3) 1 SINC (1= SINCRON OSC 1 CON FREQ OSC 3) 0 GATE (1= INIZIA ATTACK/ DECAY/SUSTAIN 0=INIZIA RELEASE
SIDREG5	D405	54277	VOCE 1 ATTACK/DECAY 7-4 ATTACK (0-15) 3-0 DECAY (0-15)

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
		REGISTRI SID	
SIDSIDREG6	D406	54278	VOCE 1 SUSTAIN/RELEASE
			7-4 SUSTAIN (0-15) 3-0 RELEASE (0-15)
SIDREG7	D407	54279	VOCE 2 FREQ BASSA
SIDREG8	D408	54280	VOCE 2 FREQ ALTA
SIDREG9	D409	54281	VOCE 2 LARGH D'IMPULSO BASSA
SIDREG10	D40A	54282	VOCE 2 LARGH D'IMPULSO ALTA (0-15)
SIDREG11	D40B	54283	VOCE 2 REG DI CONTROLLO
			7 RUMORE (1= RUMORE) 6 IMPULSO (1= IMPULSO) 5 SEGA (1= DENTE DI SEGA) 4 TRI (1= TRIANGOLO 3 TEST (1= DISABILITA OSCILLAT) 2 SUONO (1= MODUL SUONO OSC 2 CON OUTPUT OSC 1) 1 SINC (1= SINCRONIZ OSC 2 CON FREQ OSC 2) 0 GATE (1= INIZIA ATTACK/ DECAY/ SUSTAIN 0=INIZIA RELEASE
SIDREG12	D40C	54284	VOCE 2 ATTACK DECAY
			7-4 ATTACK (0-15) 3-0 DECAY (0-15)
SIDREG13	D40D	54285	VOCE 2 SUSTAIN/ RELEASE
			7-4 SUSTAIN (0-15) 3-0 RELEASE (0-15)
SIDREG14	D40E	54286	VOCE 3 FREQ BASSA
SIDREG15	D40F	54287	VOCE 3 FREQ ALTA
SIDREG16	D410	54288	VOCE 3 LARGH D'IMPULSO BASSA
SIDREG17	D411	54289	VOCE 3 LARGH D'IMPULSO ALTA (0-15)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
		REGISTRI SID	
SIDREG18	**D412**	**54290**	**VOCE 3 REG DI CONTROLLO**
			7 RUMORE (1= RUMORE)
			6 IMPULSO (1= IMPULSO)
			5 SEGA (1= DENTE DI SEGA)
			4 TRI (1= TRIANGOLO)
			3 TEST (1= DISABILITA OSCIL-LAT)
			2 SUONO (1= MODUL SUONO OSC 3 CON OUT-PUT OSC 2)
			1 SINC (1= SINCRONIZ OSC 3 CON FREQ OSC 2)
			0 GATE (1= INIZIA ATTACK/ DE-CAY/ SUSTAIN 0= INIZIA RELEA-SE)
SIDREG19	**D413**	**54291**	**VOCE 3 ATTACK DECAY**
			7-4 ATTACK (0-15)
			3-0 DECAY (0-15)
SIDREG20	**D414**	**54292**	**VOCE 3 SUSTAIN/RELEASE**
			7-4 SUSTAIN (0-15)
			3-0 RELEASE (0-15)
SIDREG21	**D415**	**54293**	**FREQ DI CUTOFF FILTRO BASSA**
SIDREG22	**D416**	**54294**	**FREQ DI CUTOFF FILTRO ALTA**
SIDREG23	**D417**	**54295**	**RISONANZA/FILTRO**
			7-4 RISON DEL FILTRO (0-15)
			3 INPUT ESTERNO FILTRO (1 = SI)
			2 OUTPUT VOCE 3 FILTRO (1 = SI)
			1 OUTPUT FILTRO VOCE 2 (1 = SI)
			0 OUTPUT FILTRO VOCE 1 (1 = SI)

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
			REGISTRI SID
SIDREG24	D418	54296	MODO/VOLUME
			7 — OUTPUT DI CUTOFF VOCE 3 (1=OFF)
			6 — SELEZIONA FILTRO PASSA ALTO (1=ON)
			5 — SELEZIONA FILTRO PASSA BANDA (1=ON)
			4 — SELEZIONA FILTRO PASSA BASSO (1=ON)
			3-0 — OUTPUT DEL VOLUME (0-15)
SIDREG25	D419	54297	POT X,CONV A/D,PADDLE 1
SIDREG26	D41A	54298	POT Y,CONV A/D,PADDLE 2
SIDREG27	D41B	54299	OSC 3, GENERATORE NUM A CASO
SIDREG28	D41C	54300	OUTPUT DEL GENERATORE ENVELOPE 3

UNITÀ DI GESTIONE DELLA MEMORIA DEL C128, REGISTRI PRIMARI IMPLEMENTI DEI MODI C128, C64, & CP/M 3.0

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
MMUCR1	D500	54528	REGISTRO DI CONFIGURAZ
PCRA	D501	54529	REG A DI PRECONFIG
PCRB	D502	54530	REG B DI PRECONFIG
PCRC	D503	54531	REG C DI PRECONFIG
PCRD	D504	54532	REG D DI PRECONFIG BIT ($D500-$D504)
			7-6 — BANCO RAM (0-3)
			5-4 — ROM ALTA (SYSTEM, INT,EXT,RAM)
			3-2 — ROM MID (SYSTEM, INT,EXT,RAM)
			1 — ROM BASSA (SYSTEM, RAM)
			0 — I/O (I/O BLOCCO,O ROM ALTA)
MMUMCR	D505	54533	REG MODO CONFIGURAZ
			7 — SENSO TASTO 40/80
			6 — OS MODO, 0=C128/1=C64
			5 — SENSO LINEA/EXROM
			4 — SENSO LINEA/GAME
			3 — FSDIR
			2-1 — NON USATO
			0 — PROCESSORE, 0=Z80/1=8502

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE

UNITÀ DI GESTIONE DELLA MEMORIA DEL C128, REGISTRI PRIMARI IMPLEMENTI DEI MODI C128, C64 & CP/M 3.0

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
MMURCR	D506	54534	REGISTRO DI CONFIG RAM 7-6 BANCO VIC RAM (VA17 & VA16) 5-4 BLOCCO RAM (PER FUTURE ESPANS) 3-2 STATO RAM SHARE (NESSUNO, BOT,TOP, ENTRAMBI) 1-0 QUANTIT RAM SHARE (1K,4K,8K 16K)
MMUP0L	D507	54535	PAG 0 PUNTAT BASSO
MMUP0H	D508	54536	PAG 0 PUNTAT ALTO
MMUP1L	D509	54537	PAG 1 PUNTAT BASSO
MMUP1H	D50A	54538	PAG 1 PUNTAT ALTO

SCAMBIANO LA PAGINA 0 E/O LA PAGINA 1 CON QUALSIASI ALTRA PAGINA NELLO SPAZIO DI INDIRIZZO 256K

BIT ($D508 & $D50A)

7-4	—
3-2	A19-A18 (USATI NEL SIST 1MB)
1-0	A17-A16 (SISTEMA 256K)

BIT ($D507 & $D509)

7-0	A15-A8

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
MMUVER	D50B	54539	NUMERO DI VERSIONE MMU 7-4 VERSIONE BANCO 3-0 VERSIONE MMU

CONTROLLORE DEL VIDEO C128 AD 80 COLONNE

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
VDCADR	D600	54784	REG DI INDIR 8563

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE

BIT DEI REGISTRI DI INDIRIZZO E DEI DATI

	7	6	5	4	3	2	1	0
D600——WRITE	—	—	R5	R4	R3	R2	R1	R0
READ	STATO	LP	VBLANK	—	—	—	—	—
VDCAT	D601		54785	REG DATI 8563				
D601——	D7	D6	D5	D4	D3	D2	D1	D0

DATA REGISTRI ADDIZIONALI 8563 NON SONO VISIBILI PER L'8502

	D700	55040	BLOCCO I/O RISERVATO
VICCOL	D800	55296	MATRICE COL VIC,1KB ($D800-$DBFF)

CIA#1 6526, ADATTATORE #1 AD INTERFACCIA COMPLESSA PER TASTIERA, JOYSTICK, PADDLE, PENNA OTTICA, DISCO VELOCE

D1PRA	DC00	56320	PORTA A (COLONNE DI OUTPUT TASTIERA)

BIT ($DC00)

0	PRA0: KEYBD O/P C0/JOY DIREZIONE #1
1	PRA1: KEYBD O/P C1/JOY DIREZIONE #1
2	PRA2: KEYBD O/P C2/JOY DIREZIONE #1/TASTO DI FUOCO PADDLE
3	PRA3: KEYBD O/P C3/JOY DIREZIONE #1/TASTO DI FUOCO PADDLE
4	PRA4: KEYBD O/P C4/JOY TASTO DI FUOCO #1
5	PRA5: KEYBD O/P C5/
6	PRA6: KEYBD O/P C6/ /SELEZ PADDLE DELLA PORTA #1
7	PRA7: KEYBD O/P C7/ /SELEZ PADDLE DELLA PORTA #2

D1PRB	DC01	56321	PORTA B (RIGHE DI INPUT TASTIERA)

BIT ($DC01)

0	PRB0: KEYBD I/P R0/JOY DIREZIONE #2
1	PRB1: KEYBD I/P R1/JOY DIREZIONE #2/ TASTO DI FUOCO PADDLE
2	PRB2: KEYBD I/P R2/JOY DIREZIONE #2/ TASTO DI FUOCO PADDLE
3	PRB3: KEYBD I/P R3/JOY DIREZIONE #2/
4	PRB4: KEYBD I/P R4/JOY TASTO DI FUOCO #2
5	PRB5: KEYBD I/P R5/
6	PRB6: KEYBD I/P R6/TIMER B: OUTPUT DI IMPULSO/LEVETTA
7	PRB7: KEYBD I/P R7/TIMER A: OUTPUT DI IMPULSO/LEVETTA

D1DDRA	DC02	56322	DIREZ DATI PORTA A
D1DDRB	DC03	56323	DIREZ DATI PORTA B
D1T1L	DC04	56324	TA BASSO

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE

CIA#1 6526, ADATTATORE #1 AD INTERFACCIA COMPLESSA PER TASTIERA, JOYSTICK, PADDLE, PENNA OTTICA, DISCO VELOCE

D1T1H	DC05	56325	TA ALTO (TIMER A)
D1T2L	DC06	56326	TB BASSO
D1T2H	DC07	56327	TB ALTO (TIMER B)
D1TOD1	DC08	56328	TOD (DECIMI)
D1TODS	DC09	56329	TOD (SECONDI)
D1TODM	DC0A	56330	TOD (MINUTI)
D1TODH	DC0B	56331	TOD (ORE)
D1SDR	DC0C	56332	REG DATO SERIALE
D1ICR	DC0D	56333	REG DI CONTROLLO INTERRUZIONE
D1CRA	DC0E	56334	REGISTRO DI CONTROLLO A
D1CRB	DC0F	56335	REGISTRO DI CONTROLLO B

CIA#1 6526 ADATTATORE AD INTERFACCIA COMPLESSA #2 INGRESSO DELL'UTENTE, RS-232, BUS SERIALE, MEMORIA VIC, NMI

D2PRA	DD00	56576	PORTA A,BUS SERIALE,RS- 232,VA14 & VA15

BIT

0	PRA0: VA14
1	PRA1: VA15
2	PRA2: RS232 OUTPUT DATI
3	PRA3: ATN OUTPUT SERIALE
4	PRA4: CLK OUTPUT SERIALE
5	PRA5: OUTPUT DATI SERIALE
6	PRA6: CLK INPUT SERIALE
7	PRA7: INPUT DATO SERIALE

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
D2PRB	DD01	56577	PORTA B,INGR UTENTE,RS-232 BIT 0 PRB0:INGR UTENTE/RS-232 DATO RICEVUTO 1 PRB1:INGR UTENTE/RS-232 INVIO RICHIESTA 2 PRB2:INGR UTENTE/RS-232 DATO TERMINALE PRONTO 3 PRB3:INGR UTENTE/RS-232 INDIC SUONO 4 PRB4:INGR UTENTE/RS-232 RIVELA PORTATORE CARICHE 5 PRB5:INGR UTENTE 6 PRB6:INGR UTENTE/RS-232 AZZERA PER INVIO 7 PRB7:INGR UTENTE/RS-232 DATI PRONTI
D2DDRA	DD02	56578	PORTA A DIREZ DATI
D2DDRB	DD03	56579	PORTA B DIREZ DATI
D2T1L	DD04	56580	TA BASSO
D2T1H	DD05	56581	TA ALTO (TIMER A)
D2T2L	DD06	56582	TB BASSO
D2T2H	DD07	56583	TB ALTO (TIMER B)
D2TOD1	DD08	56584	TOD (DECIMI)
D2TODS	DD09	56585	TOD (SECONDI)
D2TODM	DD0A	56586	TOD (MINUTI)
D2TODH	DD0B	56587	TOD (ORE)
D2SDR	DD0C	56588	REG DATI SERIALE
D2ICR	DD0D	56589	REGISTRI DI INTERRUZIONE CONTROLLO (NMI)
D2CRA	DD0E	56590	REG A DI CONTROLLO
D2CRB	DD0F	56591	REG B DI CONTROLLO
IO1	DE00	56832	SLOT DI I/O DI ESPANS (RISERVATO)
IO2	DF00	57088	SLOT DI I/O DI ESPANS (RISERVATO) CONTROLLORE DMA C128 PER ACCESSO RAM DI ESPANS UNITÀ OPZIONALE NELLA MAPPA DEL BLOCCO IO2 VIA PORTA DI ESPANS SYSTEM (PRELIM)
DMA ST	DF00	57088	REG DI STATO DEL CONTROLLORE DMA (SOLO LETTURA)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
			7 INTERRUZ IN SOSPESO (1= INT ASPETTA DI ESSERE SERVITA) 6 FINE BLOCCO (1=TRASFER COMPLETO) 5 FAULT (1= ERRORE VERIFICA BLOCCO) 4 MISURA (0= ESP MEM= 128K) (1= ESP MEM= 512K) 3-0 VERSIONE
DMA CMD	DF01	57089	REG DMA DI COMANDO DEL CONTROLLORE
			7 ESEGUE 6 RISERVATO 5 CARICA (1= ABIL AUTO=CARICAM) 4 $FF00 (1=DISABIL $FF00 DECADE) 3 RISERVATO 2 RISERVATO 1-0 MODO (00=SPOSTA DA INTERNO AD ESTERNO, 01=DA EST A INT, 10=SWAP 11=VERIF)
DMA ADL	DF02	57090	LSB DI INDIR INTERNO (C128) PER L'ACCESSO
DMA ADH	DF03	57091	MSB DI INDIR INTERNO (C128) PER L'ACCESSO
DMA LO	DF04	57092	LSB DI ESPAN INTERNA RAM PER L'ACCESSO
DMA HI	DF05	57093	MSB DI ESPAN ESTERNA RAM PER L'ACCESSO
DMA BNK	DF06	57094	64K BIT BANCO RAM ESTERNA ($DF06)

Mappa della memoria del C128 (continua)

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
			7-3 NON USATO 2-0 NUM BANCO DI ESPANS
DMA DAL	DF07	57095	LSB DEL BYTE CONTO
DMA DAH	DF08	57096	MSB DEL BYTE CONTO (CONTO DEL BLOCCO)
DMA SUM	DF09	57097	REG MASCHERAM DI INTERRUZ
			7 ABIL INTERR (1=INTERRUZ ABILITATA) 6 FINE MASCHERAM BLOCCO (1= INTERRUZ A FINE BLOCCO) 5 VERIFICA ERRORE (1= INTERRUZ DI VERIFICA ERRORE)
DMA VER	DF0A	57098	REG DI CONTROLLO INDIRIZZO BIT 7 E 6 0,0=INCREM ENTRAMBI INDIRIZZI (DEFAULT) 0,1=FISSA IND ESPANSIONE 1,0=FISSA IND C128 1,1=FISSA ENTRAMBI INDIRIZZI
	E000	57344	ROM KERNAL (8K SISTEMA OPERATIVO, $E000-$FFFF)

REGISTRI SECONDARI MMU

ETICHETTA INDIR. DI MEMORIA	INDIR. ESADEC.	INDIR. DEC.	DESCRIZIONE
MMUCR	FF00	65280	REGISTRO DI CONFIGURAZ (SECONDARIO)
LCRA	FF01	65281	REG A DI CARICAM DELLA CONFIGURAZIONE
LCRB	FF02	65282	REG B DI CARICAM DELLA CONFIGURAZIONE
LCRC	FF03	65283	REG C DI CARICAM DELLA CONFIGURAZIONE
LCRD	FF04	65284	REG D DI CARICAM DELLA CONFIGURAZIONE BIT ($FF00-$FF04)
			7-6 BANCO RAM (0-3) 5-4 ROM ALTA (SYSTEM, INT,EXT, RAM) 3-2 ROM MEDIA (SYSTEM, INT,EXT, RAM) 1 ROM BASSA (SYSTEM, RAM) 0 I/O (I/O O ROM ALTA)

TABELLA DI SALTO KERNAL

NUOVE ENTRATE PER IL C128

JMP SPIN SPOUT	FF47	65351	PONE PORTA SERIALE VELOCE PER I/O
JMP CLOSE ALL	FF4A	65354	CHIUDE TUTTI I FILE LOGICI DI UN'UNITÀ
JMP C64 MODE	FF4D	65357	RICONFIG IL SISTEMA COME C64 (NO RITORNO)
JMP DMA CALL	FF50	65360	INIZ RICHIESTA DMA PER ESPANS RAM ESTERNA, INVIA COMANDO A UNITÀ DMA
JMP BOOT CALL	FF53	65363	BOOT PROGRAMMA DI CARICAM DA DISCHETTO
JMP PHOENIX	FF56	65366	CHIAMA TUTTE ROUTINE CARD DI FUNZIONI A PARTENZA FREDDA, INIZIAL
JMP LKUPLA	FF59	65369	CERCA TAB PER LA DEFINITO
JMP LKUPSA	FF5C	65372	CERCA TAB PER SA DEFINITO
JMP SWAPPER	FF5F	65375	PASSA TRA 40 E 80 COLONNE (EDITOR)
JMP DLCHR	FF62	65378	INIZ RAM AD 80 COLONNE (EDITOR)
JMP PFKEY	FF65	65381	TASTO FUNZIONI DI PROGRAMMA (EDITOR)
JMP SETBNK	FF68	65384	PONE BANCO PER OPERAZIONI DI I/O
JMP GETCFG	FF6B	65387	CERCA DATI MMU PER IL BANCO DEFINITO
JMP JSRFAR	FF6E	65390	JSR IN QUALS BANCO, RTS AL BANCO RICHIAMATO
JMP JMPFAR	FF71	65393	JMP IN QUALS BANCO
JMP INDFET	FF74	65396	LDA (FETVEC), Y DA QUALS BANCO
JMP INDSTA	FF77	65499	STA (STAVEC), Y IN QUALS BANCO
JMP INDCMP	FF7A	65402	CMP (CMPVEC), Y IN QUALS BANCO
JMP PRIMM	FF7D	65405	STAMPA UTILITY IMMEDIATE (JSR SEMPRE IN QUESTA ROUTINE)

TABELLA DI SALTO KERNAL STANDARD

	FF80	65408	LIBERA NUM DEL KERNAL
JMP CINT	FF81	65409	INIZ CHIP EDITOR & VIDEO (EDITOR)

Mappa della memoria del C128 (continua)

TABELLA DI SALTO KERNAL STANDARD

JMP IOINIT	FF84		65412	INIZ UNITÀ DI I/O (PORTE, TIMER, ETC,)
JMP RAMTAS	FF87		65415	INIZIAL RAM E BUFFER PER IL SISTEMA
JMP RESTOR	FF8A		65418	RIPRISTINA I VETTORI PER INIZIAL SISTEM
JMP VECTOR	FF8D		65421	CAMBIA VETTORI PER L'UTENTE
JMP SETMSG	FF90		65424	CONTROLLA IL MESSAGGIO O.S.
JMP SECND	FF93		65472	INVIA SA DOPO LISTEN
JMP TKSA	FF96		65430	INVIA SA DOPO TALK
JMP MEMTOP	FF99		65433	PONE/LEGGE LA CIMA SISTEMA RAM
JMP MEMBOT	FF9C		65436	PONE/LEGGE BASSO DEL SISTEMA RAM
JMP KEY	FF9F		65439	ANAL TASTIERA (EDITOR)
JMP SETTMO	FFA2		65442	PONE TIMOUT IN IEEE (RISERVATO)
JMP ACPTR	FFA5		65445	HANDSHAKE DEL BYTE SERIALE IN
JMP CIOUT	FFA8		65448	HANDSHAKE DEL BYTE SERIALE OUT
JMP UNTLK	FFAB		65451	INVIA UNTALK SERIALE OUT
JMP UNLSN	FFAE		65454	INVIA UNLISTEN SERIALE OUT
JMP LISTN	FFB1		65457	INVIA LISTEN SERIALE OUT
JMP TALK	FFB4		65460	INVIA TALK SERIALE OUT
JMP READSS	FFB7		65463	RIPORTA IL BYTE DI STATO DI I/O
JMP SETLFS	FFBA		65460	PONE LA, FA, SA
JMP SETNAM	FFBD		65469	PONE LUNGHEZZA E NOME DEL FILE DELL'INDIRIZZO
JMP (IOPEN)	FFC0	OPEN	65472	APRE IL FILE LOGICO
JMP (ICLOSE)	FFC3	CLOSE	65475	CHIUDE IL FILE LOGICO
JMP (ICHKIN)	FFC6	CHKIN	65478	PONE CANALE IN
JMP (ICKOUT)	FFC9	CKOUT	65481	PONE OUT IL CANALE
JMP (ICLRCH)	FFCC	CLRCH	65484	RIPRISTINA IL CANALE DI I/O PER DEFAULT
JMP (IBASIN)	FFCF	BASIN	65487	INPUT DAL CANALE
JMP (IBSOUT)	FFD2	BSOUT	65490	OUTPUT VERSO IL CANALE
JMP LOADSP	FFD5		65493	CARICA DA FILE
JMP SAVESP	FFD8		65496	SALVA NEL FILE
JMP SETTIM	FFDB		65599	PONE IL CLOCK INTERNO
JMP RDTIM	FFDE		65502	LEGGE IL CLOCK INTERNO
JMP (ISTOP)	FFE1	STOP	65505	ANALIZZA IL TASTO
JMP (IGETIN)	FFE4	GETIN	65508	LEGGE I DATI BUFFERIZZATI
JMP (ICLALL)	FFE7	CLALL	65511	CHIUDE TUTTI I FILE E CANALI

TABELLA DI SALTO KERNAL STANDARD

JMP UDTIM	**FFEA**	**CLOCK**	**65514**	**INCREMENTA CLOCK INTERNO**
JMP SCRORG	**FFED**		**65517**	**RIPORTA LA LARGH DELLA FINESTRA DELLO SCHERMO (EDITOR)**
JMP PLOT	**FFF0**		**65520**	**LEGGE/PONE X,Y COORD CURSORE (EDITOR)**
JMP IOBASE	**FFF3**		**65523**	**RIPORTA BASE DI I/O**
SYSTEM	**FFF8**		**65528**	**VETTORE DEL SISTEMA OPERAT (RAM1)**
NMI	**FFFA**		**65530**	**VETTORE PROCESSORE NMI**
RESET	**FFFC**		**65532**	**VETTORE PROCESSORE DI RESET**
IRQ	**FFFE**		**65534**	**VETTORE PROCESSORE IRQ/BRK**

FLAG EDITOR/KERNAL E REGISTRI AUSILIARI

I simboli seguenti sono usati dall'Editor del C128. Notate che il gestore dello schermo VIC Editor IRQ dipende da questi. Nella maggior parte dei casi i contenuti di queste locazioni saranno posti direttamente nel registro appropriato e dovranno essere usati al posto del registro attuale. Per esempio, per modificare la locazione dell'insieme dei caratteri usati dal VIC, usate VM1 ($0A2C) invece del registro 24 VIC ($D018). VM1 sarà usato dall'editor per aggiornare il registro 24 del VIC.

INDIRIZ./NOME	SPIEGAZIONE
$00D8/GRAPHM	**VEDERE SOTTO. SE=$FF ALLORA L'EDITOR LASCIA IL VIC SOLO.**
$00D9/CHAREN	**MASCHERA PER I BIT 8502/CHAREN.**
$0A2C/VM1	**MATRICE VIDEO DEL MODO TESTO VIC & PUNTATORE DEL CARATTERE DI BASE.**
$0A2D/VM2	**MATRICE VIDEO DEL MODO GRAFICO VIC & PUNTATORE DEL BIT MAP.**
$0A2E/VM3	**INDIRIZZO DI BASE DEL VIDEO TESTO 8563.**
$0A2F/VM4	**INDIRIZZO DI BASE DEGLI ATTRIBUTI 8563.**
$0A34/SPLIT	**IN MODO DIVISIONE DELLO SCHERMO, CONTIENE IL VALORE PER L'IRQ DEL RASTER DI MEZZO.**
$0A2B/CURMOD	**MODO CURSORE 8563.**
$0A21/PAUSE	**FLAG S DI CONTROLLO (IN REALTÀ = $13)**

SPIEGAZIONE DI VARI BYTE DI FLAG EDITOR/KERNAL, ETC.

SIMBOLO DI INDIRIZZO		DESCRIZIONE							
		7	6	5	4	3	2	1	0
0000	D6510	—	(IN)	(OUT)	(IN)	(OUT)	(OUT)	(OUT)	(OUT)
0001	R6510	—	CAPKEY	CASMTR	CASSEN	CASWRT	CHAREN	HIRAM	LORAM
00F7	LOCKS	CASE	CTL S	—	—	—	—	—	—
00F8	SCROLL	OFF	LINKER						
00D3	SHFLAG	—	—	—	ALT	ALPHA	CTRL		SHIFT
0A22	RPTFLG	ALL	NONE	—	—	—	—	—	—
0A26	BLNON	ON	BLNK	—	—	—	—	—	—
00F9	BEEPER	ON	—	—	—	—	—	—	—
00D8	GRAPHM	MCM	SPLIT	BMM	—	—	—	—	—
00D7	MODE	40/80	—	—	—	—	—	—	—
0A04	INIT_ STATUS	CHRSET	CINT	—	—	—	—	—	BASIC

Note sui Simboli Kernal:

Init-Status. Consultare anche parte sopra. Permette al sistema di sapere cosa è stato inizializzato e cosa non lo è stato. Posto in $00 dà un reset ma non viene toccato da NMI.

System-Vector. Dove va il Kernal se deve andare da qualche parte. Viene posto in BASIC freddo al reset. Il BASIC stesso lo pone in BASIC caldo dopo che lo ha inizializzato. Anche il monitor rispetta quest'operazione.

System. Vettore nella RAM1 in $FFF8. Pone un inserimento in MODO C128, l'utente lo può dirigere nuovamente al suo codice. Preso al momento del reset fornendo sempre all'utente il controllo (protezione) dal reset.

MAPPA DELLA MEMORIA DEL COMMODORE 64

ETICHETTA	INDIR. ESADEC.	LOCAZ. DEC.	DESCRIZIONE
D6510	0000	0	Reg 6510 On-Chip Direz-Dati
R6510	0001	1	Reg 6510 On- Chip 8 Bit di Input/ Output
	0002	2	Non usato
ADRAY1	0003-0004	3-4	Vettore di Salto: cambia Mobile in Intero
ADRAY2	0005-0006	5-6	Vettore di Salto: cambia Intero in Mobile

ETICHETTA	INDIR. ESADEC.	LOCAZ. DEC.	DESCRIZIONE
CHARAC	0007	7	Cerca Carat
ENDCHR	0008	8	Flag: analizza Frase a Fine Stringa
TRMPOS	0009	9	Colonna dello Schermo Dalla Ultima TAB
VERCK	000A	10	Flag: 0=Carica 1=Verifica
COUNT	000B	11	Puntatore al Buffer di Input/Num di Subscritti
DIMFLG	000C	12	Flag: DIMens Array per Default
VALTYP	000D	13	Tipo di Dato: $FF=Stringa, $00=Numerico
INTFLG	000E	14	Tipo di Dato: $80=Intero, $00=Mobile
GARBFL	000F	15	Flag: anal DATO LIST frase/Garbage Coll
SUBFLG	0010	16	Flag: Rif Subscritto/Chiamata della Funzione Utente
INPFLG	0011	17	Flag: $00=INPUT $40=GET,$98=READ
TANSGN	0012	18	Flag: segno TAN Compara Risult 0013 19 Flag: Frase di INPUT
LINNUM	0014-0015	20-21	Temp: Valore Intero
TEMPPT	0016	22	Puntatore: Stack della Stringa Temporanea
LASTPT	0017-0018	23-24	Indir della Ultima Stringa Temporanea
TEMPST	0019-0021	25-33	Stack per le Stringhe Temporanee
INDEX	0022-0025	34-37	Area del Puntatore di Servizio
RESHO	0026-002A	38-42	Prodotto a Virgola Mobile della Moltipl.
TXTTAB	002B-002C	43-44	Puntatore: Iniz Testo in BASIC
VARTAB	002D-002E	45-46	Puntatore: Iniz Variabili in BASIC
ARYTAB	002F-0030	47-48	Puntatore: Iniz Matrici BASIC
STREND	0031-0032	49-50	Puntatore: Fine Matrici BASIC (+1)
FRETOP	0033-0034	51-52	Puntatore: Base Memorizz. Stringhe
FRESPC	0035-0036	53-54	Puntatore Stringa di Servizio
MEMSIZ	0037-0038	55-56	Puntatore: Ind più alto usato dal BASIC
CURLIN	0039-003A	57-58	Numero Linea Corrente in BASIC
OLDLIN	003B-003C	59-60	Numero Linea Preced in BASIC
OLDTXT	003D-003E	61-62	Puntatore: Frase BASIC per CONT
DATLIN	003F-0040	63-64	Numero della Linea Corrente DATA
DATPTR	0041-0042	65-66	Puntatore: Ind Item Corrente DATA
INPPTR	0043-0044	67-68	Vettore: Routine di INPUT
VARNAM	0045-0046	69-70	Nome Variabile Corrente BASIC
VARPNT	0047-0048	71-72	Puntatore: Dato Della Variabile Corrente BASIC
FORPNT	0049-004A	73-74	Puntatore: Var Indice per FOR/NEXT
	004B-0060	75-96	Puntatore Temp Area Dati
FACEXP	0061	97	Accumulatore a Virgola Mobile #1: Esponente

La mappa della memoria del Commodore 64 (continua)

ETICHETTA	INDIR. ESADEC.	LOCAZ. DEC.	DESCRIZIONE
FACHO	0062-0065	98-101	Accum a Virg Mob #1: Mantissa
FACSGN	0066	102	Accum a Virg Mob #1: Segno
SGNFLG	0067	103	Puntatore: Costante Serie di Valutazione
BITS	0068	104	Accum a Virg Mob #1: Overflow della Cifra
ARGEXP	0069	105	Accum a Virg Mob #2: Esponente
ARGHO	006A-006D	106-109	Accum a Virg Mob #2: Mantissa
ARGSGN	006E	110	Accum a Virg Mobile #2: Segno
ARISGN	006F	111	Risultato di Comparaz del Segno: Accum #1 contro #2
FACOV	0070	112	Accum a Virg Mob #1: Ordine Basso (Arrotondato)
FBUFPT	0071-0072	113-114	Puntatore: Buffer della Cassetta
CHRGET	0073-008A	115-138	Subroutine: prende Byte Seguente del Testo BASIC
CHRGOT	0079	121	Entrata per prendere lo stesso Byte di Testo una seconda volta
TXTPTR	007A-007B	122-123	Puntatore Byte Corrente del Testo BASIC
RNDX	008B-008F	139-143	Valore Seme Della Funz RND a Virg Mobile
STATUS	0090	144	Parola di I/O di Stato Kernal: ST
STKEY	0091	145	Flag: tasto STOP/tasto RVS
SVTX	0092	146	Tempo Costante per Registraz
VERCK	0093	147	Flag: 0=Caricam 1=Verifica
C3PO	0094	148	Flag: Bus Seriale-Car di Output Bufferizzati
BSOUR	0095	149	Car Bufferiz per Bus Seriale
SYNO	0096	150	Num Sinc Cassetta
	0097	151	Area Temp Dati
LDTND	0098	152	Num di File Aperti/Indice Tab dei File
DFLTN	0099	153	Unità Input per Default (0)
DFLTO	009A	154	Unità Output (CMD) per Default (3)
PRTY	009B	155	Registraz di Car di Parità
DPSW	009C	156	Flag: Registr Byte Ricevuto
MSGFLG	009D	157	Flag: $80=Modo Diretto, $00=Programma
PTR1	009E	158	Passo 1 del Nastro Errore Log
PTR2	009F	159	Passo 2 del Nastro Errore Log
TIME	00A0-00A2	160-162	Clock Jiffy in Tempo Reale (appros) 1/60 Sec
	00A3-00A4	163-164	Area Temp Dati
CNTDN	00A5	165	Cuntdown del Sinc della Cassetta
BUFPNT	00A6	166	Puntatore: I/O del Buffer del Nastro
INBIT	00A7	167	Bit di Input RS-232/Temp Cassetta
BITCI	00A8	168	Conto Bit di Input RS-232/Temp Cassetta

ETICHETTA	INDIR. ESADEC.	LOCAZ. DEC.	DESCRIZIONE
RINONE	00A9	169	Flag RS-232: Controlla Bit di Partenza
RIDATA	00AA	170	Buffer Byte di Input RS-232/Temp Cassetta
RIPRTY	00AB	171	Input di Parità Rs-232/Cnt Breve Cassetta
SAL	00AC-00AD	172-173	Puntatore: Buffer del Nastro/Scroll dello Schermo
EAL	00AE-00AF	174-175	Indir Fine Nastro/Fine Programma
CMPO	00B0-00B1	176-177	Costanti di Misura del Tempo del Nastro
TAPE1	00B2-00B3	178-179	Puntatore: Iniz Buffer del Nastro
BITTS	00B4	180	Conteggio Bit Out RS-232/Temp Cassetta
NXTBIT	00B5	181	RS-232 Pross ! Bit da Inviare/Flag Nastro EOT
RODATA	00B6	182	RS-232 Buffer del Byte Out
FNLEN	00B7	183	Lungh Nome File Corrente
LA	00B8	184	Num File Log Corrente
SA	00B9	185	Ind Sec Corrente
FA	00BA	186	Num Unità Corrente
FNADR	00BB-00BC	187-188	Puntatore: Nome File Corrente
ROPRTY	00BD	189	RS-232 Par Out/Temp Cassetta
FSBLK	00BE	190	Legge Cassetta/Scrive Cont Blocco
MYCH	00BF	191	Buffer Parola Seriale
CAS1	00C0	192	Blocco Motore Nastro
STAL	00C1-00C2	193-194	I/O dell'Ind di Partenza
MEMUSS	00C3-00C4	195-196	Temp Caricam Nastro
LSTX	00C5	197	Tasto Corr Premuto: CHR$(n)0= no Tasto
NDX	00C6	198	Num di Car nel Buffer Tastiera (Coda)
RVS	00C7	199	Flag: Stampa Car Inverso-1=Si, 0=Non Usato
INDX	00C8	200	Puntatore: Fine Linea Logica per INPUT
LXSP	00C9-00CA	201-202	Pos Cursore X-Y ad Iniz INPUT
SFDX	00CB	203	Flag: Stampa Car Spostati
BLNSW	00CC	204	Abil Lampeg Cursore:0=Flash del Cursore
BLNCT	00CD	205	Timer: Count down Cursore Bistabile
GDBLN	00CE	206	Car sotto Cursore
BLNON	00CF	207	Flag: Lampeg Ultimo Curs On/Off
CRSW	00D0	208	Flag: INPUT o GET da tastiera
PNT	00D1-00D2	209-210	Puntatore: Ind Linea Corrente Schermo
PNTR	00D3	211	Col Cursore su Linea Corr
QTSW	00D4	212	Flag: Editor Modo Frase, $00=NO
LNMX	00D5	213	Lungh Linea Schermo Fisico

La mappa della memoria del Commodore 64 (continua)

ETICHETTA	INDIR. ESADEC.	LOCAZ. DEC.	DESCRIZIONE
TBLX	00D6	214	Num Linea Fisica del Cursore Corrente
	00D7	215	Area Temp Dati
INSRT	00D8	216	Flag: Modo Inserim, >0=#INST
LDTB1	00D9-00F2	217-242	Tab Link Linea Schermo/Temp Editor
USER	00F3-00F4	243-244	Puntatore: RAM Loc Col Schermo Corr
KEYTAB	00F5-00F6	245-246	Vettore: Tab Decodif di Tastiera
RIBUF	00F7-00F8	247-248	RS-232 Puntat Buffer di Input
ROBUF	00F9-00FA	249-250	RS-232 Puntat Buffer di Output
FREKZP	00FB-00FE	251-254	Spazio Libero Pag 0 per Programmi Utente
BASZPT	00FF	255	Area Temp Dati BASIC
	0100-01FF	256-511	Area Stack Microproces del Sistema
	0100-010A	256-266	Floating nell'Area di Operaz della Stringa
BAD	0100-013E	256-318	Errore Log Input Nastro
BUF	0200-0258	512-600	Buffer di INPUT del Sistema
LAT	0259-0262	601-610	Tab KERNAL: Num Attivi del File Log
FAT	0263-026C	611-620	Tab KERNAL: Num Unità di ogni File
SAT	026D-0276	621-630	Tab KERNAL: Ind Sec per ogni File
KEYD	0277-0280	631-640	Coda Buffer Tastiera (FIFO)
MEMSTR	0281-0282	641-642	Puntatore: Basso Memoria per O.S.
MEMSIZ	0283-0284	643-644	Puntatore: Cima Memoria per O.S.
TIMOUT	0285	645	Flag: Variab Kernal per Timeout IEEE
COLOR	0286	646	Codice Col del Carattere Corrente
GDCOL	0287	647	Col di Fondo Sotto Cursore
HIBASE	0288	648	Alto Memoria Schermo (Pag)
XMAX	0289	649	Largh Buffer Tastiera
RPTFLG	028A	650	Flag: Tasto REPEAT usato, $80= Ripeti
KOUNT	028B	651	Contatore Ripetiz di Velocità
DELAY	028C	652	Contatore Ripetiz di Ritardo
SHFLAG	028D	653	Flag: Tasto SHIFT Tastiera/Tasto CTRL/Tasto
LSTSHF	028E	654	Ultima Forma Shift Tastiera
KEYLOG	028F-0290	655-656	Vettore: Posiz Tab Tastiera
MODE	0291	657	Flag: $00=Disabil Tasti SHIFT, $80=Abil Tasti SHIFT
AUTODN	0292	658	Flag: Auto Scroll Giu 0=ON
M51CTR	0293	659	RS-232: Immagine Reg di Controllo 6551
M51CDR	0294	660	RS-232: Immagine Reg di Comando 6551

ETICHETTA	INDIR. ESADEC.	LOCAZ. DEC.	DESCRIZIONE
M51AJB	0295-0296	661-662	RS-232 BPS non Standard (Tempo/2-100) USA
RSSTAT	0297	663	RS-232: Immag Reg di Stato 6551
BITNUM	0298	664	RS-232 Num di Bit da Inviare
BAUDOF	0299-029A	665-666	RS-232 Baud Rate: Bit a Tempo Pieno
RIDBE	029B	667	RS-232 Indice a Fine Buffer di Input
RIDBS	029C	668	RS-232 Iniz Buffer di Input (Pag)
RODBS	029D	669	RS-232 Iniz Buffer di Output (Pag)
RODBE	029E	670	RS-232 Indice a Fine Buffer di Output
IRQTMP	029F-02AO	671-672	Tiene Vettore IRQ Durante I/O Nastro
ENABL	02A1	673	Abil RS-232
	02A2	674	Senso TOD durante I/O Cassetta
	02A3	675	Temp di Mem per Lettura Cassetta
	02A4	676	Temp D1IRQ Indicat per Lettura di Cassetta
	02A5	677	Temp per Indice Linea
	02A6	678	PAL/NTSC Flag 0=NTSC,1=PAL
	02A7-02FF	697-767	Non Usato
IERROR	0300-0301	768-769	Vettore: Stampa Messag Errore BASIC
IMAIN	0302-0303	770-771	Vettore: Iniz Caldo BASIC
ICRNCH	0304-0305	772-773	Vettore: Tokenize Testo BASIC
IQPLOP	0306-0307	774-775	Vettore: LISTA Testo BASIC
IGONE	0308-0309	776-777	Vettore: Smista Car BASIC
IEVAL	030A-030B	778-779	Vettore: Val Segno BASIC
SAREG	030C	780	Memoriz per Reg .A 6502
SXREG	030D	781	Memoriz per Reg .X 6502
SYREG	030E	782	Memoriz per Reg .Y 6502
SPREG	030F	783	Mem per Reg .SP 6502
USRPOK	0310	784	Indstr USR Funzione di Salto (4C)
USRADD	0311-0312	785-786	Byte Ind Basso USR/Byte Alto
	0313	787	Non Usato
CINV	0314-0315	788-789	Vettore: Int Irq Hardware
CBINV	0316-0317	790-791	Vettore: Int Instr BRK
NMINV	0318-0319	792-793	Vettore: Int Non Mascher
IOPEN	031A-031B	794-795	Vettore Rout KERNAL OPEN
ICLOSE	031C-031D	796-797	Vettore Rout KERNAL CLOSE
ICHKIN	031E-031F	798-799	Vettore Rout KERNAL CHKIN
ICKOUT	0320-0321	800-801	Vettore Rout KERNAL CHKOUT
ICLRCH	0322-0323	802-803	Vettore Rout KERNAL CLRCHN
IBASIN	0324-0325	804-805	Vettore Rout KERNAL CHRIN
IBSOUT	0326-0327	806-807	Vettore Rout KERNAL CHROUT
ISTOP	0328-0329	808-809	Vettore Rout KERNAL STOP
IGETIN	032A-032B	810-811	Vettore Rout KERNAL GETIN
ICLALL	032C-032D	812-813	Vettore Rout KERNAL CLALL
USRCMD	032E-032F	814-815	Vettore def dall'Utente

La mappa della memoria del Commodore 64 (continua)

ETICHETTA	INDIR. ESADEC.	LOCAZ. DEC.	DESCRIZIONE
ILOAD	0330-0331	816-817	Vettore Rout KERNAL LOAD
ISAVE	0332-0333	818-819	Vettore Rout KERNAL SAVE
	0334-033B	820-827	Non Usato
TBUFFR	033C-03FB	828-1019	Buffer I/O del Nastro
	03FC-03FF	1020-1023	Non Usato
VICSCN	0400-07FF	1024-2047	Area Memoria Schermo 1024 Byte
	0400-07E7	1024-2023	Matrice Video 25 Linee per 40 Colonne
	07F8-07FF	2040-2047	Puntatore Dati Sprite
	0800-9FFF	2048-40959	Spazio Progr Norm BASIC
	8000-9FFF	32768-40959	Cartuccia ROM VSP-8192 Byte
	A000-BFFF	40960-49151	ROM BASIC-BFFF Byte (o 8K RAM)
	C000-CFFF	49152-53247	RAM-4096 Byte
	D000-DFFF	53248-57343	Unità di I/O e Colore RAM o Generat Car ROM o RAM-4096 Byte
	E000-FFFF	57344-65535	ROM KERNAL-8192 Byte (o 8K RAM)

ASSEGNAZIONI INPUT/OUTPUT DEL COMMODORE 64

ESADEC.	DEC.	BIT	DESCRIZIONE
0000	0	7-0	MOS 6510 Reg Direz Dati (xx101111) Bit=1:Output,Bit=0: Input,x=Entrambi
0001	1		MOS 6510 Porta I/O Microprocess su Chip
		0	/Segnale LORAM (0= Spegne BASIC ROM)
		1	/Segnale HIRAM (0= Spegne Kernal ROM)
		2	Segnale CHAREN (0= Accende Car ROM)
		3	Linea Output Dati della Cassetta
		4	Senso Commutatore della Cassetta 1=Commutat Chiuso
		5	Controllo Motore della Cassetta 0=ON, 1=OFF
		6-7	Non Definiti

ESADEC.	DEC.	BIT	DESCRIZIONE
D000-D02E	53248-54271		MOS 6566 CONTROLLORE INTERFACCIA VIDEO (VIC)
D000	53248		Sprite 0 Pos X
D001	53249		Sprite 0 Pos Y
D002	53250		Sprite 1 Pos X
D003	53251		Sprite 1 Pos Y
D004	53252		Sprite 2 Pos X
D005	53253		Sprite 2 Pos Y
D006	53254		Sprite 3 Pos X
D007	53255		Sprite 3 Pos Y
D008	53256		Sprite 4 Pos X
D009	53257		Sprite 4 Pos Y
D00A	53258		Sprite 5 Pos X
D00B	53259		Sprite 5 Pos Y
D00C	53260		Sprite 6 Pos X
D00D	53261		Sprite 6 Pos Y
D00E	53262		Sprite 7 Pos X
D00F	53263		Sprite 7 Pos Y
D010	53264		Sprite 0-7 Pos X (msb di coord X)
D011	53265		Reg di Controllo VIC
		7	Compara Raster: (8 bit) Vedi 53266
		6	Modo Testo Colore Esteso: 1=Abilitato
		5	Modo Bit-Map:1=Abil
		4	Schermo Azzerato al Colore del Bordo: 0=Blank
		3	Selez Video Testo 24/25 Righe:1= 25 Righe
		2-0	Scroll Dolce alla Posizione Y Dot (0-7)
D012	53266		Legge Raster/Scrive Val Raster per IRQ Comparato
D013	53267		Latch Penna Ottica Pos X
D014	53268		Latch Penna Ottica Pos Y
D015	53269		Abilita Video Sprite: 1=abilitato
D016	53270		Reg di Controllo VIC
		7-6	Non Usato
		5	PONE QUESTO BIT SEMPRE A 0
		4	Modo Multicolore: 1=Abilita (Testo o Bit Map)
		3	Seleziona Video Testo 38/40 Col: 1=40 Colonne
		2-0	Scroll Dolce alla Posiz X
D017	53271		Sprite 0-7 Espande 2 per Verticale (Y)
D018	53272	7-4	Indir Base Matrice Video (dentro VIC)

Assegnazioni Input/Output del Commodore 64 (continua)

ESADEC.	DEC.	BIT	DESCRIZIONE
		3-1	Indir di Base Car Dot-Data (dentro il VIC)
D019	53273		Reg di Interruz Flag VIC (Bit=1: Capita IRQ)
		7	Pone Qualsiasi Condizione di IRQ VIC Abilitato
		3	Flag Penna Ottica Triggerata IRQ
		2	Flag IRQ di Collis Sprite contro Sprite
		1	Flag IRQ di Collis Sprite contro Fondo
		0	Flag IRQ di Comparaz Raster
D01A	53274		Reg di Mascheram IRQ: 1= Interruzione Abilitata
D01B	53275		Priorità Video Sprite contro Fondo 1=Sprite
D01C	53276		Selez Modo Multicol Sprite 0-7: 1=M.C.M.
D01D	53277		Sprite 0-7 Espande 2 per Oriz (X)
D01E	53278		Rileva Collisione Sprite contro Sprite
D01F	53279		Rileva Collisione Sprite contro Sfondo
D020	53280		Colore del Bordo
D021	53281		Colore di Fondo 0
D022	53282		Colore di Fondo 1
D023	53283		Colore di Fondo 2
D024	53284		Colore di Fondo 3
D025	53285		Reg Sprite Multi Colore 0
D026	53286		Reg Sprite Multi Colore 1
D027	53287		Colore Sprite 0
D028	53288		Colore Sprite 1
D029	53289		Colore Sprite 2
D02A	53290		Colore Sprite 3
D02B	53291		Colore Sprite 4
D02C	53292		Colore Sprite 5
D02D	53293		Colore Sprite 6
D02E	53294		Colore Sprite 7
D400-D7FF	54272-55295		UNITÀ INTERFACCIA SOUND MOS 6581 (SID)
D400	54272		Voce 1: Controllo Frequenz-Byte Basso
D401	54273		Voce 1: Controllo Frequenz-Byte Alto
D402	54274		Voce 1: Ampiez Forma d'Onda Impuls-Byte Basso
D403	54275	7-4	Non Usato
		3-0	Voce 1: Ampiez Forma d'Onda Impuls-Nybble Alto

ESADEC.	DEC.	BIT	DESCRIZIONE
D404	54276		Voce 1: Reg di Controllo
		7	Selez Forma d'Onda Rumore a caso, 1=On
		6	Selez Forma d'Onda Impuls, 1=On
		5	Selez Forma d'Onda Dente di Sega, 1=On
		4	Selez Forma d'Onda Triangolo, 1=On
		3	Test Bit: 1=Disabil Oscillatore 1
		2	Modula Suono Output Osc 1 con Osc 3, 1=On
		1	Sincr Frequenza di Osc 1 con Osc 3, 1=On
		0	Gate Bit: 1=Inizia Att/Dec/Sus, 0=Iniz Release
D405	54277		Generatore Envelope 1: Ciclo di Control Attack/Decay
		7-4	Selez durata Ciclo Attack: 0-15
		3-0	Selez durata Ciclo Dacey: 0-15
D406	54278		Generatore Envelope 1: Controllo Ciclo Sustain/Release
		7-4	Selez durata Ciclo Sustain: 0-15
		3-0	Selez durata Ciclo Release: 0-15
D407	54279		Voce 2: Controllo Frequenza-Byte Basso
D40B	54280		Voce 2: Controllo Frequenza-Byte Alto
D409	54281		Voce 2: Ampiez Forma d'Onda Impuls-Byte Basso
D40A	54282	7-4	Non Usato
		3-0	Voce 2: Ampiez Forma d'Onda Impuls-Nybble Alto
D40B	54283		Voce 2: Reg di Contr
		7	Selez Forma d'Onda Rumore a caso, 1=On
		6	Selez Forma d'Onda Impuls, 1=On
		5	Selez Forma d'Onda Dente di Sega, 1=On
		4	Selez Forma d'Onda Triangolo, 1=On
		3	Test Bit: 1=Disabil Oscillatore 2
		2	Modula Suono Output Osc 2 con Osc 1, 1=On
		1	Sincr Freq Osc 2 con Osc 1, 1=On
		0	Gate Bit: 1=Inizia Att/Dec/Sus, 0=Iniz Release
D40C	54284		Generatore Envelope 2: Controllo Ciclo Attack/Decay
		7-4	Selez durata Ciclo Attack: 0-15

Assegnazioni Input/Output del Commodore 64 (continua)

ESADEC.	DEC.	BIT	DESCRIZIONE
		3-0	Selez durata Ciclo Decay: 0-15
D40D	54285		Generatore Envelope 2: Controllo Ciclo Sustain/Release
		7-4	Selez durata Ciclo Sustain: 0-15
		3-0	Selez durata Ciclo Release: 0-15
D40E	54286		Voce 3: Controllo Freq-Byte Basso
D40F	54287		Voce 3: Controllo Freq-Byte Alto
D410	54288		Voce 3: Ampiez Forma d'Onda Impuls-Byte Basso
D411	54289	7-4	Non Usato
		3-0	Voce 3: Ampiez Forma d'Onda-Nybble Alto
D412	54290		Voce 3: Reg di Controllo
		7	Selez Forma d'Onda Rumore a caso, 1=On
		6	Selez Forma d'Onda Impuls, 1=On
		5	Selez Forma d'Onda Dente di Sega, 1=On
		4	Selez Forma d'Onda Triangolo, 1=On
		3	Test Bit: 1=Disabil Oscillatore 3
		2	Modula Suono Output Osc 3 con Osc 2, 1=On
		1	Sincr Freq Osc 3 con Osc 2, 1=On
		0	Gate Bit: 1=Inizia Att/Dec/Sus, 0=Iniz Release
D413	54291		Generatore Envelope 3: Controllo Ciclo Attack/Decay
		7-4	Selez durata Ciclo Attack: 0-15
		3-0	Selez durata Ciclo Decay: 0-15
D414	54292		Generatore Envelope 3: Controllo Ciclo Sustain//Release
		7-4	Selez durata Ciclo Sustain: 0-15
		3-0	Selez durata Ciclo Release: 0-15
D415	54293		Freq Cutoff del Filtro: Nybble Basso (Bit 2-0)
D416	54294		Freq Cutoff del Filtro: Byte Alto
D417	54295		Controllo Risonanza Filtro/Controllo Input della Voce
		7-4	Selez Rison Filtro: 0-15
		3	Input Filtro Ester: 1=Si, 0=No
		2	Output Voce 3 Filtro: 1=Si, 0=No
		1	Output Voce 2 Filtro: 1=Si, 0=No

ESADEC.	DEC.	BIT	DESCRIZIONE
		0	Output Voce 1 Filtro: 1=Si, 0=No
D418	54296		Selez Modo Filtro e Volume
		7	Output Cut-Off Voce 3: 1=Off, 0=On
		6	Selez Modo Filtro Passa Alto: 1=On
		5	Selez Modo Filtro Passa Banda: 1=On
		4	Selez Modo Filtro Passa Basso: 1=On
		3-0	Selez Output Volume: 0-15
D419	54297		Convertitore Analog Digitale: Paddle Game 1 (0-255)
D41A	54298		Convertitore Analog Digitale: Paddle Game 2 (0-255)
D41B	54299		Generat Num a caso Oscillatore 3
D41C	54300		Output Generatore Envelope 3
D500-D7FF	54528-55285		IMMAGINI SID
D800-DBFF	55296-56319		Colore RAM (Nybble)
DC00-DCFF	56320-56575		Adattatore ad Interfaccia Comples MOS 6526 (CIA)#1
DC00	56320		Dati della Porta A (Tastiera, Joystick, Paddle, Penna Ottica)
		7-0	Valori di Scrittura Colon Tastiera per Analisi Tastiera
		7-6	Legge Paddle su Porta A/B (01=Porta A, 10=Porta B)
		4	Bottone di Fuoco Joystick A: 1=Fuoco
		3-2	Tasti di Fuoco del Paddle
		3-0	Direz A Joystick (0-15)
DC01	56321		Dati della Porta B (Tastiera, Joystick, Paddle): Porta Game 1
		7-0	Valori Lettura Righe Tastiera per Analisi Tastiera
		7	Timer B:Output di Levetta/Impulso
		6	Timer A: Output di Levetta/Impulso
		4	Tasto Fuoco del Joystick 1: 1=Fuoco
		3-2	Tasti di Fuoco del Paddle
		3-0	Direz 1 Joystick
DC02	56322		Reg Direz dei Dati-Porta A (56320)
DC03	56323		Reg Direz dei Dati-Porta B (56321)
DC04	56324		Timer A: Byte Basso
DC05	56325		Timer A: Byte Alto
DC06	56326		Timer B: Byte Basso

Assegnazioni Input/Output del Commodore 64 (continua)

ESADEC.	DEC.	BIT	DESCRIZIONE
DC07	65327		Timer B: Byte Alto
DC08	65328		Clock Ora del Giorno: 1/10 Secondi
DC09	56329		Clock Ora del Giorno: Secondi
DC0A	56330		Clock Ora del Giorno: Minuti
DC0B	56331		Clock Ora del Giorno: Ore + Flag AM/PM (Bit 7)
DC0C	56332		Buffer Dati I/O Seriale Sincrono
DC0D	56333		Reg di Controllo Interruz CIA (Legge IRQ/Scrive Maschera)
		7	Flag IRQ (1=Capita IRQ)/Pone-Cancella il Flag
		4	IRQ FLAG1 (Legge Cassetta/Input SRQ Bus Seriale)
		3	Interruz Porta Seriale
		2	Interruz Segnale Clock Ora del Giorno
		1	Interruz Timer B
		0	Interruz Timer A
DC0E	56334		Reg di Controllo A CIA
		7	Freq Clock Ora del Giorno: 1=50 Hz, 0=60 Hz
		6	Modo I/O Porta Seriale: 1=Output 0=Input
		5	Conteggi Timer A: 1= Segnali CNT, 0=Clock del Sistema 02
		4	Timer di Caricam Forzato A: 1=Si
		3	Modo Esegui Timer A: 1=un Colpo, 0=Continuo
		2	Modo Output Timer A verso PB6: 1=Levetta 0=Impulso
		1	Output Timer A verso PB6: 1=Si, 0=No
		0	Parte/Ferma Timer A 1=Parte, 0=Ferma
DC0F	56335		Reg di Controllo B CIA
		7	Pone Allarme/TOD Clock: 1=Allarme, 0=Clock
		6-5	Selez Modo Timer B: 00=Conto Impulsi Sistema 02 Clock 01=Conto Transiz Positive CNT 10=Conto Impulsi Underflow del Timer A 11=Conto Underflow del Timer A con CNT Positivo
		4-0	Come Reg di Control A CIA-per Timer B
DD00-DDFF	56576-56831		Adattatore della Interf Complessa MOS 6526 #2 (CIA)

ESADEC.	DEC.	BIT	DESCRIZIONE
DD00	56576		Dati della Porta A (Bus Seriale, RS-232, Controllo Memoria VIC)
		7	Input Dati del Bus Seriale
		6	Input Impulso del Clock del Bus Seriale
		5	Output Dati del Bus Seriale
		4	Output Impulso del Clock del Bus Seriale
		3	Output del Segnale ATN del Bus Seriale
		2	Output Dati RS-232 (Ingresso Utente)
		1-0	Selez Banco di Mem del Sistema VIC Chip (Default=11)
DD01	56577		Dati della Porta B (Ingresso Utente, RS-232)
		7	Utente/Insieme Dati RS-232 Pronto
		6	Utente/RS-232 azzera per l'Invio
		5	Utente
		4	Utente/Riconosce il Portatore di Cariche
		3	Utente/Indicatore Suono RS-232
		2	Utente/Terminale Dati Pronto RS-232
		1	Utente/Richiesta da Inviare RS-232
		0	Utente/Dati Ricevuti RS-232
DD02	56578		Reg Direz Dati-Porta A
DD03	56579		Reg Direz Dati-Porta B
DD04	56580		Timer A: Byte Basso
DD05	56581		Timer A: Byte Alto
DD06	56582		Timer B: Byte Basso
DD07	56583		Timer B: Byte Alto
DD08	56584		Clock Ora del Giorno: 1/10 Secondi
DD09	56585		Clock Ora del Giorno: Secondi
DD0A	56586		Clock Ora del Giorno: Minuti
DD0B	56587		Clock Ora del Giorno: Ore+Flag AM/PM (Bit 7)
DD0C	56588		Buffer Dati I/O Sincrono Seriale
DD0D	56589		Reg CIA Controllo Interruzione (Legge NMI/Scrive Maschera)
		7	Flag NMI (1=NMI Capitato)/Pone-Azzera Flag
		4	FLAG1 NMI (Input Dati Ricevuti Utente/RS-232)
		3	Interruz Porta Seriale
		1	Interruz Timer B
		0	Interruz Timer A
DD0E	56590		Reg Controllo A CIA
		7	Freq Clock Ora del Giorno: 1=50 Hz, 0=60 Hz
		6	Modo I/O Porta Seriale: 1=Output 0=Input

Assegnazioni Input/Output del Commodore 64 (continua)

ESADEC.	DEC.	BIT	DESCRIZIONE
		5	Conto Timer A: 1=Segnali CNT, 0=Clock Sistema 02
		4	Caricam Forzato Timer A: 1=Si
		3	Modo Esecuzione Timer A: 1=Un Colpo, 0=Continuo
		2	Modo Output del Timer A verso PB6: 1=Levetta, 0=Impulso
		1	Output Timer A su PB6: 1=Si, 0=No
		0	Iniz/Ferma Timer A: 1=Iniz, 0=Ferma
DD0F	56591		Reg Controllo B CIA
		7	Pone Allarme/Clock TOD: 1=Allarme, 0=Clock
		6-5	Selez Modo Timer B: 00=Conto Impulso del Sistema Clock 02 01=Conto Transiz dei CNT Positivi 10=Conto Impulsi Underflow Timer A 11=Conto dello Underflow Timer A quando CNT è Positivo
		4-0	Come Reg di Controllo CIA A-per Timer B
DE00-DEFF	56832-57087		Riservato per Espans I/O Future
DF00-DFFF	57088-57343		Riservato per Espans I/O Future

17

SPECIFICHE HARDWARE DEL C128

Questo capitolo descrive l'hardware del Commodore C128, i requisiti del circuito integrato VLSI ed il rapporto tra la configurazione dell'hardware ed il sistema operativo.

Il personal computer C128 è compatibile con il software e le unità periferiche del C64. Oltre alla compatibilità col C64, esiste un modo C128 nel quale 128K della RAM sono disponibili per l'uso del sistema/utente. La versione in BASIC 7.0 è il linguaggio per default. Il Kernal Commodore viene fornito in forma compatibile.

Il C128 ha anche un coprocessore Z80 che può fare pieno uso delle utility della RAM del sistema e del Kernal, che si usano con i sistemi operativi potenti come la versione 3.0 del CP/M. Lo schema della ROM in banchi permette a tasti di funzioni software di essere installati internamente al sistema o aggiunti esternamente come cartuccia di espansione.

Un'altra grossa funzione è la capacità video ad 80 colonne dell'8563, disponibile in modo C128 come aggiunta al modo a 40 colonne.

Un supporto di unità periferiche include l'interfaccia Commodore per mouse, joystick, paddle, penna ottica (entrambe le penne ottiche a 40 ed 80 colonne); il Datassette Commodore; l'Ingresso dell'Utente, che sopporta l'RS232; i modem; il Bus di Espansione, che sopporta l'espansione esterna della memoria ed il Bus Seriale standard Commodore, che sopporta tutte le componenti esistenti del Bus Seriale. Ci sono anche parecchie funzioni che riducono l'appesantimento del software, come la pagina zero rilocabile e lo stack del sistema. In modo C64, sono disponibili i 66 tasti standard. In modo C128, sono disponibili altri 26 tasti, inclusi i tasti separati del cursore, un tasto **HELP,** ulteriori tasti di funzioni ed un vero tasto **CAPS LOCK.** I tasti aggiuntivi, raggruppati nella tastiera alternativa, sono definibili da parte dell'utente e ciò aumenta la flessibilità e la comunicatività del sistema con l'utente.

Ecco un riassunto delle funzioni del C128:

- Compatibilità col C64
- Capacità video di 80 colonne
- Processore Z80 (versione CP/M 3.0 (2 MHz))
- Operazione 2 MHz 8502 in modo 80 colonne
- 128K del sistema standard RAM
- 48K del sistema standard ROM
- 32K della funzione interna della ROM (facoltativi)
- 32K della cartuccia esterna ROM
- Interfaccia del disk drive seriale veloce
- Tastiera completa, 92 tasti con il tasto **CAPS LOCK,** il tasto **HELP** e tasti separati per il controllo del cursore

ARCHITETTURA DEL SISTEMA

Il computer C128 utilizza una struttura condivisa del bus simile a quella del C64. Il bus condiviso emula la porta duale RAM e ROM, che permette ai caratteri ROM, al colore della RAM ed al sistema RAM di essere utilizzati sia dal microprocessore che dal controllore video VIC 8564, senza interfaccia tra di loro. Questo richiede che la RAM sia abbastanza veloce per fornire dati validi in metà tempo di un ciclo macchina normale del microprocessore. Questa condivisione normale è controllata da un coprocessore che abiliterà o disabiliterà il processore alternativamente a metà del ciclo macchina.

Il sistema C128 divide il bus di indirizzo in sezioni condivise e non condivise. Tutte le normali parti di I/O dell'8502 sono nel bus di indirizzo non condiviso; il chip VIC ed i suoi chip di supporto associati sono localizzati nel bus condiviso. Il chip VIC porterà in circuito gli indirizzi del processore nel bus condiviso basato sulla sua linea di controllo AEC. I dati del bus sono in comune ad entrambi i lati del bus di indirizzo.

Il processore interfaccia con la maggior parte dei chip del sistema come fosse un ciclo bus standard 6502, dove un ciclo macchina è uguale ad un ciclo clock. Questo permette di usare parti da 1 MHz per un clock da 1 MHz ed esclude la necessità di creare indirizzi e dati validi di impulsi stroboscopici, poichè queste informazioni vengono ora date dai lati del clock principale, Φ0. Selezioni chip per l'I/O e per la ROM del sistema sono generate dal PLA che traccia gli indirizzi del microprocessore durante Φ1.

Per gli accessi alla RAM del sistema, l'indirizzo della riga è l'indirizzo dal microprocessore e l'indirizzo della colonna è l'indirizzo di output MMU (chiamato Indirizzo Tradotto). Gli output dell'**Indirizzo Tradotto** vengono calcolati considerando i contenuti del Registro della Configurazione MMU ed il Registro della Configurazione RAM. Da questi valori, l'MMU genera sia indirizzi normali che tradotti, una selezione CAS. I circuiti elettronici CAS—gating nell'MMU abilitano uno dei due banchi di 64K di RAM in un sistema di 128K. Per il ciclo di accesso VIC della RAM viene posto il banco RAM in modo indipendente dal banco del processore. Una scrittura nella ROM risulterà nella RAM sottostante, mentre una lettura dalla ROM disabiliterà sempre CAS in entrambi i banchi. MMU permette adattamenti della RAM in entrambi i banchi 0 e 1 fatti per il cliente.

La formazione di banchi nella ROM viene effettuata posizionando bit nel Registro di Configurazione contenuto nell'MMU e comunicato al decodificatore PLA. Questo permette che la Funzione Banking della ROM e qualsiasi cartuccia del C128 siano incluse nella configurazione del sistema base.

PLA genera selezioni chip per il colore RAM, registri di controllo VIC e caratteri della ROM, che vengono usati durante i cicli VIC e del processore, come tutte le selezioni di cui c'era bisogno per la ROM solo—processore e per le unità periferiche. Per evitare la conflittualità dei bus, PLA deve generare anche disabilitazioni CAS per qualsiasi accesso alla ROM o ad unità di I/O.

Il chip VIC genera i segnali usati per controllare la Memoria Dinamica e fornisce funzioni di macro controllo come il ricaricamento della RAM. Lo scopo

primario del VIC è di recuperare i dati dello schermo dalla memoria, usando sia una condivisione di cicli che il DMA e di creare un output video NTSC — o PAL — compatibile che viene applicato ad un monitor o modulato ed applicato ad un apparecchio TV. Il C128 fornisce output per output video Composti, Cromo/Luminosi e RF dal chip VIC, come anche un latch di input a margine triggerato della penna ottica diretto al chip VIC.

L'output dal generatore del suono del chip SID è bufferizzato e applicato direttamente ad un amplificatore esterno, come quello che si trova in un monitor esterno, o modulato e riprodotto nell'apparecchio televisivo dell'utente. Il chip SID ha anche un input esterno per miscelare una seconda sorgente del suono.

Il chip di controllo video dell'8563 va a prendere i dati da una sezione particolare della RAM a cui ci si riferisce come video RAM e crea un output RGBI (Red — Green — Blue — Intensity) per l'uso di un monitor esterno ad 80 colonne. L'8563 crea anche tutti i segnali di cui avete bisogno per una ricarica dinamica della sua RAM video particolare. Il C128 fornisce output RGBI e monocromatici composti dal chip 8563.

La porta della cassetta viene implementata utilizzando le porte della pagina zero disponibili sull'8502 ed il controllo software dell'handshaking dell'hardware. La porta del bus seriale Commodore viene implementata in modo simile usando un CIA 6526 per l'I/O. Il bus seriale funziona con componenti seriali Commodore ed in modo C64 viene pilotato concretamente dalle routine software contenute nel C64. L'Ingresso dell'Utente è una porta multiuso che comprende molte linee parallele di porte che sopportano le unità periferiche come un RS232 lento, il modem, ecc. Le porte del joystick sono identiche a quelle sul C64 e vengono implementate usando un CIA 6526 per leggere/scrivere sulla porta.

Il connettore video ha un video composto e output separati per il colore e la luminosità per un uso con i monitor. I monitor Commodore 1701 e 1702 interfacciano direttamente con questo connettore. Il connettore femmina (jack) RF di output fornisce un segnale RF compatibile con le regolazioni delle unità che interfacciano con la TV ed è selezionabile con un interruttore tra i canali 3 e 4. Sono sopportati entrambi gli standard televisivi NTSC e PAL. Il connettore RGBI ed il segnale sono simili a quelli usati dall'IBM e sono compatibili con la maggior parte dei monitor che sopportano l'RGBI di Tipo I. Inoltre, è disponibile un segnale monocromatico composto sul connettore RGBI ed è generalmente compatibile con NTSC (o PAL) composto. L'audio è disponibile solo dal connettore video/audio a 40 colonne.

SPECIFICHE DEL SISTEMA

Tratta funzioni e limiti del sistema C128, incluse le descrizioni del sistema, le sue configurazioni e fattori limitanti come potenza, caricamento e operazioni.

La Figura 17-1 mostra il sistema C128.

Figura 17-1. Il Sistema C128

ARCHITETTURA DEL BUS DEL SISTEMA

I bus descritti includono:

- Bus processore
- Bus di indirizzo tradotto
- Bus di indirizzo Multiplex
- Bus di indirizzo condiviso
- Bus del dato colore
- Bus video

IL BUS PROCESSORE

Il **Bus Processore** include i bus dati ed indirizzo che sono connessi direttamente al processore 8502. Questi bus sono designati con D_0-D_7 per il bus dati a 8 bit e con A_0-A_{15} per il bus di indirizzo a 16 bit. Questi bus legano il processore con la maggior parte delle ROM del sistema e con le unità di I/O, incluse almeno parti di tutte le ROM del sistema, tutte le funzioni ROM incorporate, l'MMU, il PLA, il processore video 8563, il SID ed entrambi i chip CIA.

Il bus processore è anche in comunicazione con il coprocessore Z80. Tutte le linee di indirizzo sono condivise direttamente da entrambi i processori. Per permettere allo Z80 di operare su una famiglia di bus 6502, è necessario trattenere l'invio dei dati nello Z80 e chiudere la strada ai dati che lasciano lo Z80. Così, lo Z80 ha un piccolo bus dati locale, designato con ZD_0-ZD_7. Durante un ciclo di scrittura, quando AEC è alto, i dati Z80 sono chiusi nel bus processore. Durante un ciclo di lettura, i dati del bus processore sono chiusi nel bus dati Z80. Questa lettura di dati è bloccata palesemente dal clock del sistema ad 1 MHz.

NOTA: I cicli di lettura e di scrittura a cui ci si riferisce in questo documento, a meno che non siano specificati in altro modo, sono cicli bus di tipo 8502. Gli output dello Z80 Abilita Lettura ed Abilita Scrittura sono condizionati dall'uso logico per interfacciarsi con un ciclo bus 8502, così non viene fatta nessuna distinzione come per le differenze tra i cicli dei diversi processori. Per ulteriori informazioni su questa logica, consultate la sezione sul processore Z80 e sulla Schematica del C128, della parte Commodore Num.310378.

Come detto prima, lo Z80 non è in comunicazione diretta con il bus dati del processore poichè è necessario adattare lo Z80 al protocollo del bus 8502. Notate, tuttavia, che qualsiasi altra unità e qualsiasi altro bus (tranne due che saranno trattati più avanti) condividono il bus dati del processore come fosse un bus dati comune.

BUS DI INDIRIZZO TRADOTTO

Un altro bus del sistema C128 è il **Bus di Indirizzo Tradotto** prodotto dall'MMU quando AEC è alto. Questo bus consiste solo di linee di indirizzo di ordine alto, designate con TA_8-TA_{15}. Queste linee riflettono l'azione dell'MMU sulle normali linee di indirizzo di ordine alto, che possono includere o no una sorta di traduzione. L'MMU può tradurre l'indirizzo di pagina zero o di pagina uno con una

normale operazione e traduce l'indirizzo Z80 da \$ 0000 a \$ 0FFF per fargli leggere il BIOS Z80. Una descrizione più completa delle traduzioni MMU si può trovare nella sezione MMU.

Di solito, il bus di indirizzo tradotto pilota indirettamente la RAM del sistema ed il chip VIC pilotando i bus di indirizzo multiplex. Esso pilota direttamente la linea 12 di indirizzo della ROM 4 del sistema per permettere la rilocazione della ROM Z80. Infine, questo bus riceve le linee di indirizzo da 8 a 15 della porta di espansione compatibile col C64.

Durante un ciclo VIC o DMA, l'MMU porta alti TA_{12}-TA_{15}, mentre TA_8-TA_{11} diventano a tre stati. Questo permette al chip VIC di pilotare TA_8-TA_{11} come indirizzi VIC VA_8-VA_{11}. Durante un ciclo esterno DMA le linee TA dell'MMU divengono tri-stati ed il bus TA, presumibilmente pilotato dalla fonte DMA, a sua volta pilota il bus di indirizzo del processore da A_8 a A_{15}. Questo permette alla fonte DMA di accedere a qualsiasi chip periferico tranne all'MMU. L'azione del VIC durante un ciclo VIC è descritta sotto.

IL BUS DI INDIRIZZO MULTIPLEX

Questa sezione descrive due bus di indirizzo in relazione tra loro, il **Bus di Indirizzo Multiplex** ed il **Bus di Indirizzo Multiplex VIC,** conosciuti rispettivamente come MA_0-MA_7 e VMA_0-VMA_7. Il bus di indirizzo multiplex VIC viene creato quando EAC è alto con il multiplessaggio (multiplexing = suddividere un canale tra diversi utilizzatori) del bus di indirizzo tradotto di ordine alto (TA_0-TA_{15}) con il bus di indirizzo del processore di ordine basso (A_0-A_7), controllato via segnale MUX. Questo bus, pilotato da un multiplatore hardware attraverso una serie di resistori, viene chiamato il Bus di Indirizzo Multiplex. Il bus di indirizzo multiplex VIC viene usato per l'accesso del processore ai registri del chip VIC. Viene usato anche per l'accesso VIC alla RAM del sistema.

Durante un ciclo VIC (AEC basso), verranno specificate le linee di indirizzo del chip VIC. Non c'è un bus di indirizzo completamente separato per gli indirizzi VIC, così esso condivide VMA_0-VMA_7 e le linee di indirizzo vengono a tre stati quando AEC è alto. La maggior parte degli indirizzi VIC escono dal chip VIC già in forma multiplex, ma due di essi, VA_6 e VA_7, non forniscono informazioni sulle colonne, poichè il chip VIC fornisce solo 14 bit di indirizzamento. I bit di indirizzo di ordine più alto VA_{14} e VA_{15} vengono da CIA-2 come nel C64. Ciò significa che il VIC fornisce VMA_0-VMA_7 completi per un accesso VIC DRAM o per una ricarica DRAM. TA_8-TA_{11} forniti dal VIC vengono usati unitamente ad un altro bus di indirizzo per i cicli di indirizzi VIC in forma non multiplex, come gli accessi ai caratteri ROM ed al colore RAM.

IL BUS DI INDIRIZZO COMUNE

Il **Bus di Indirizzo Comune** è un bus di indirizzo in forma non multiplex usato sia dal processore che dal chip VIC per comunicare con le risorse comuni, cioè i caratteri della ROM ed il colore della RAM (e indirettamente con il sistema RAM 8563). Quando AEC è alto, il bus di indirizzo comune, designato con SA_0-SA_7, viene pilotato da A_0-A_7, i bit di indirizzo del processore di ordine inferiore. I bit di ordine superiore di cui si ha bisogno vengono forniti dal bus di indirizzo tradotto, che è anche un bus di indirizzo comune. Così il processore è in grado di accedere ad entrambi gli articoli in comune.

Quando AEC è basso, gli indirizzi VIC VA_0-VA_7 (VMA_0-VMA_7) devono entrare nel bus di indirizzo comune. Poichè VA_0-VA_8 sono in realtà in forma multiplex, solo l'indirizzo della riga deve essere inviato al bus di indirizzo comune. Così gli indirizzi VIC in forma multiplex vengono bloccati in modo palese quando sia /RAS che MUX sono alti, ma vengono trattenuti e arrestati in seguito in modo che quando sono combinati con l'indirizzo della colonna, viene presentato l'indirizzo completo. I bit di indirizzo di ordine alto vengono qui forniti dal bus di indirizzo comune tradotto. Notate che il bus di indirizzo comune fornisce gli 8 bit più bassi dell'indirizzo della porta di espansione, permettendo al VIC di accedere alle cartucce e ad alcune capacità ulteriori del drive per mezzo dei chip TTL usati per pilotare il bus di indirizzo comune. Durante DMA, le linee SA, come quelle TA, vengono mandate indietro per pilotare il bus del processore. Come notato prima, questo permette ai chip delle unità periferiche, alla ROM ed alla RAM di essere raggiunte da una fonte DMA, come il modulo di espansione della RAM. Non si può accedere solo all'MMU e al controllore del video 8563 durante il DMA. Consultate la sezione sull'auto partenza della ROM nel quinto Volume, Capitolo 14 per maggiori dettagli sull'inizializzazione sia della ROM di espansione interna che di quella esterna.

IL BUS DEI DATI DEL COLORE

La RAM colore è scritta in o letta da un bus dei dati nybble chiamato **Bus dei Dati del Colore.** Quando AEC è alto, il bus dei dati del colore è connesso alla metà inferiore del bus dei dati del processore per mezzo di un interruttore analogico, permettendo al processore di avere pieno accesso alla RAM colore. Quando AEC è basso, l'interruttore viene aperto, isolando il bus dei dati del colore dal bus dei dati del processore. In questo stato, viene pilotato dal bus dei dati VIC esteso D_8-D_{11}. Poichè la RAM colore è a banchi, appare qui solo metà per volta.

IL BUS VIDEO

Il **Bus Video** è un bus locale rispetto al controllore video 8563, che consiste dell'**Indirizzo Video** (DA_0-DA_7) e del **Bus dei Dati Video** (DD_0-DD_7). Questo bus locale sopporta la RAM video dell'8563, che è completamente isolata dal resto del sistema del C128. Il bus di indirizzo video è un bus di indirizzo in forma multiplex che fornisce l'indirizzamento al DRAM video. Il bus dei dati video fornisce la comunicazione tra questo DRAM e l'8563. L'8563 fornisce anche gli impulsi stroboscopici per righe e colonne ed una ricarica dinamica di questo DRAM.

ORGANIZZAZIONE DELLA MEMORIA DEL SISTEMA

Questa sezione descrive la memoria del sistema C128. La Figura 17-2 è un diagramma dettagliato della Mappa della Memoria del C128.

Figura 17-2. Mappa della Memoria del C128 (Modo C64)

ORGANIZZAZIONE DELLA MEMORIA ROM DEL C128

Il modo C128 viene ottenuto con l'azzeramento del sistema ed è controllato da un bit nel Registro della Configurazione MMU. In modo C128, l'MMU si dichiara nella mappa della memoria del C128 in $ FF00 e nello spazio di I/O, cominciando in $ D500. L'uso dei registri MMU localizzati in $ FF00 permette di gestire la memoria senza che il blocco di I/O si accumuli in quel momento e con una perdita minima di RAM contigua. L'MMU viene completamente rimosso dalla mappa della memoria in modo C64. Tuttavia è ancora usato dall'hardware per gestire la memoria.

La Figura 17-2 presenta la mappa standard in modo C64. Alcuni dei modi alternativi vengono mostrati nella Figura 17-3. Tutti i modi C64 sono compatibili con il computer C64, poichè il C128 diventa in pratica un C64 se si trova in modo C64. I dettagli delle locazioni/operazioni del registro MMU sono trattati nel quinto Volume, Capitolo 14.

Le ROM in modo C64 assomigliano alle ROM del Commodore 64. BASIC e Kernal interni forniscono il modo C64 con il normale sistema operativo del Commodore 64 nella ROM. Questa ROM duplica realmente alcune delle ROM usate in modo C128, ma è necessario, poichè non è accessibile dal modo C128. In modo C128, sono presenti fino a 48K del sistema operativo, essendo l'esatta quantità posta dal controllo del software. Questo permette un accesso più veloce alla RAM di base eliminando le sezioni di cui non si ha bisogno del sistema operativo.

Le ROM esterne rappresentate nella mappa della memoria sono quelle usate in modo C64. Esse seguono le regole del Commodore 64 per l'inserimento nella mappa; cioè, le cartucce si pongono nell'hardware attraverso le linee /EXROM e /GAME. Le ROM esterne in modo C128 (cioè, le cartucce C128) sono organizzate come le ROM a banchi; quando il sistema viene inizializzato, tutti gli slot della ROM vengono interrogati sull'esistenza di una ROM, e sulla priorità della ROM, se esiste. Questo dà più flessibilità rispetto al metodo della sostituzione della ROM cablata, poichè le ROM Kernal e BASIC possono essere scambiate per un programma di applicazione, o per un controllo di programma esterno, o possono essere azzerate insieme. Questa manipolazione dei banchi è compiuta scrivendo nel Registro della Configurazione in locazione $ D500 0 $ FF00 nell'MMU.

L'hardware ha anche la funzione di memorizzare i valori presenti della configurazione e di forzare un caricamento del Registro di Configurazione scrivendo in uno dei LCR (Registri di Caricamento della Configurazione). Questo permette al programmatore di far sì che la ROM non appaia nel banco (per default) tutte le volte che si verifica un accesso ad un banco dove è memorizzato un dato.

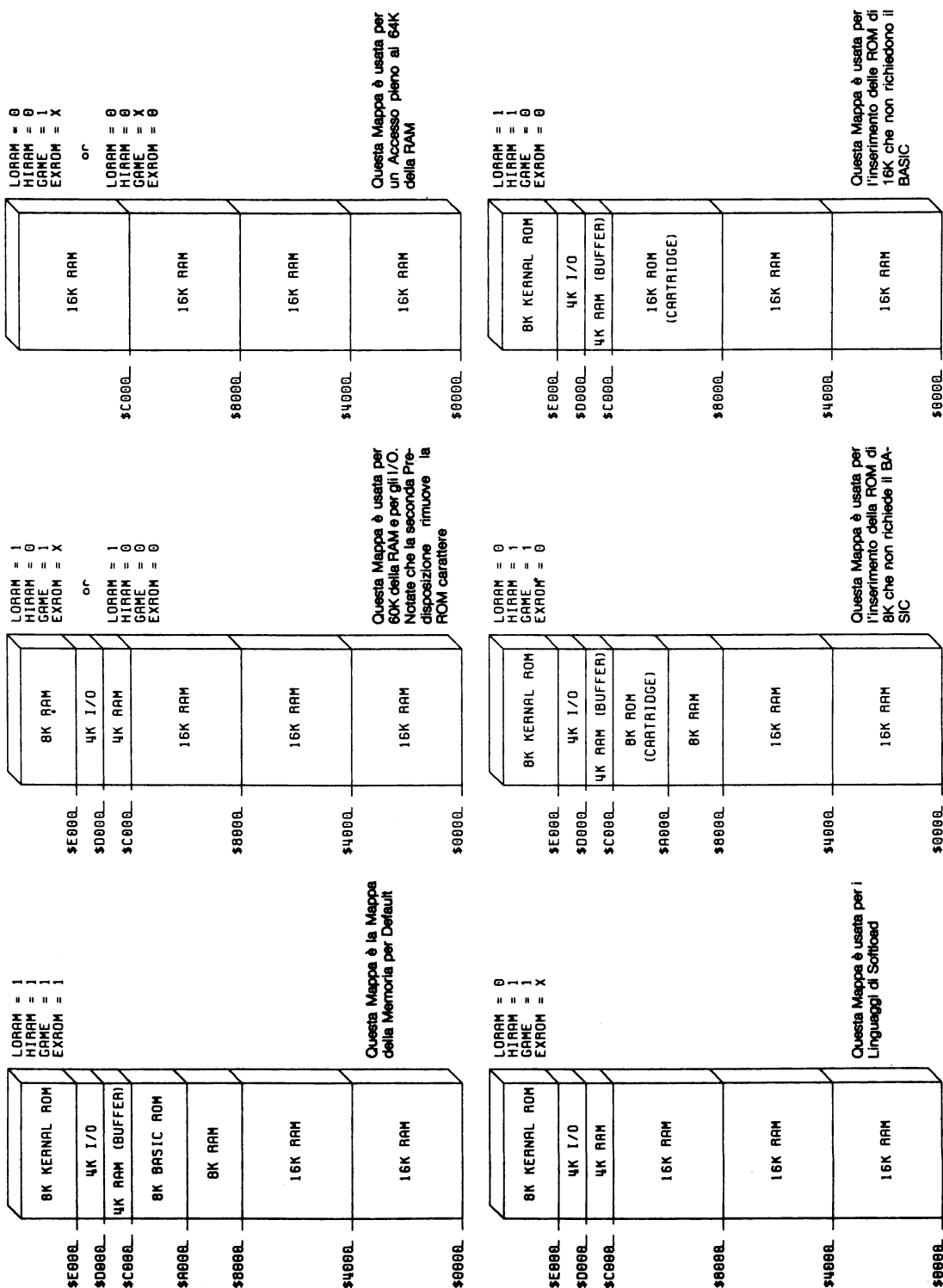



Figura 17-3. Configurazioni Alternative della Memoria del C64

ORGANIZZAZIONE DELLA MEMORIA DELLA RAM DEL C128

Come mostrato nella Figura 17-2, la RAM presente nel sistema è in realtà composta da due banchi di 64K per 8 byte di RAM. Si accede alla RAM selezionando uno dei due banchi di 64K, l'intervallo massimo di indirizzo dell'8502 e dello Z80, secondo le regole dei banchi della RAM poste nel registro di configurazione della RAM dell'MMU. Quest'area mostrata come RAM rappresenta ciò che il processore vedrebbe se tutte le ROM fossero disabilitate. Si può eseguire la commutazione tra i banchi in uno dei due modi.

Il banco in uso è una funzione del valore memorizzato nel registro di configurazione. Una memorizzazione in questo registro avrà sempre effetto immediato. Una memorizzazione indiretta in questo registro, usando valori di configurazione per un banco programmato, può essere eseguita scrivendo in uno dei registri di caricamento indiretto conosciuto come LCR, localizzato nella regione $ FF00 della memoria. Scrivendo in un LCR, i contenuti del PCR ad esso corrispondente (Registro di PreConfigurazione) saranno trattenuti nel registro di configurazione. Questo permette al programmatore di predisporre quattro diverse configurazioni preprogrammate che permettono a ciascun banco di essere personalizzato prima del tempo; cioè, il banco 1 come banco dati potrebbe essere strettamente un banco RAM senza I/O o ROM abilitati, mentre il banco 0 come banco del sistema può avere la ROM del sistema e gli I/O abilitati per default. Inoltre, la lettura di qualsiasi LCR riporterà il valore del suo PCR corrispondente.

Se trattate con banchi di 64K di memoria contemporaneamente, può essere bene accumulare nel banco 1 pur trattenendo la RAM del sistema (stack, pagina zero, schermo, ecc.). L'MMU prevede per ciò l'impiego di una «RAM **comune**». Questa RAM non accumula nei banchi ed è programmabile per misura e posizione (alto, basso, o entrambi) nella mappa della memoria. La misura viene posta dai bit 0 e 1 nel Registro della Configurazione della RAM (RCR). Se il valore dei bit è 0, 1K sarà in comune. I valori 1,2 e 3 producono aree comuni di 4K, 8K e 16K rispettivamente. Se viene posto il bit 2 dell'RCR, la memoria bassa viene tenuta in comune; se viene posto il bit 3, allora è in comune la memoria alta. In tutti i casi, la RAM comune è localizzata fisicamente nel banco 0.

La pagina zero e la pagina uno possono essere localizzate (o rilocalizzate) indipendentemente dall'RCR. Quando il processore accede ad un indirizzo che cade all'interno della pagina zero o della pagina uno, l'MMU somma all'indirizzo del processore di ordine alto i contenuti della coppia di registri P0 o P1, rispettivamente e pone questo nuovo indirizzo nel bus, includendo i bit di indirizzamento esteso A16. Si verificherà un accumulo nei banchi della RAM per accedere al nuovo indirizzo. Operazioni di scrittura nei registri P0 e P1 saranno memorizzate in prelatch finchè non si verificherà un'operazione di scrittura nel rispettivo puntatore della pagina a byte basso. Questo evita che un puntatore della pagina a byte alto P_{xH} coinvolga l'indirizzo tradotto finchè sia il byte alto che quello basso sono stati scritti. La memoria comune esclude i puntatori delle pagine.

Nello stesso tempo, i contenuti dei registri P0 e P1 sono applicati ad un confrontatore digitale e si verifica una sostituzione inversa se l'indirizzo dall'8502 cade all'interno della pagina puntata dal registro. Questo risulta in uno scambio della pagina zero o della pagina uno con la memoria che sostituiscono. Lo scambio avviene solo se l'area di scambio viene definita come RAM; cioè, il

sistema o la funzione ROM devono essere sempre nei loro indirizzi assegnati e così non dovrebbero essere sostituiti nuovamente ma naturalmente non causerebbero alcuna conflittualità. Notate che al momento del reset del sistema, i puntatori sono posti alla pagina zero reale ed alla pagina uno reale.

NOTA: Ci sono in realtà molti modi della memoria che escludono parti del banco come sono qui selezionati. Questi modi sono menzionati sotto e sono trattati nei dettagli in una sezione successiva.

Per l'accesso al chip VIC, un bit nel registro di stato MMU sostituisce la linea di indirizzo esteso A_{16}, selezionando l'abilitazione del giusto CAS per rendere possibile manovrare il VIC ovunque nell'intervallo di 128K. Notate che AEC è il meccanismo che MMU usa per manovrare uno spazio dell'indirizzo VIC; cioè, quando AEC è basso, viene assunto un accesso VIC. Questo risulta nel fatto che il banco VIC viene selezionato per un DMA esterno, poichè anche questo porterà la linea AEC all'MMU basso.

SPECIFICHE DELLE PRESTAZIONI

POTENZA ASSORBITA

La Tabella 17-1 contiene i valori della potenza assorbita del C128, includendo varie opzioni aggiunte.

I.C.		I_{Typ} (5v)	I_{Max}	I_{Typ} (9v)	I_{Max}	I_{Typ} (12v)	I_{Max}	Unità
Totale di Gruppo	**61**	**2.63**	**3.83**	**0.03**	**0.05**	**0**	**0**	**A**
UNITÀ PERIFERICHE								
Scrivania Magica	**1**	**0.24**	**0.35**	**0**	**0**	**0**	**0**	**A**
RS-232	**1**	**0.07**	**0.10**	**0**	**0**	**0**	**0**	**A**
Auto Modem	**1**	**0**	**0**	**0.07**	**0.10**	**0**	**0**	**A**
Cassetta	**1**	**0**	**0**	**0.28**	**0.40**	**0**	**0**	**A**
TOTALE DELLE UNITÀ PERIFERICHE	**4**	**0.31**	**0.45**	**0.35**	**0.50**	**0**	**0**	**A**
C128 E UNITÀ PERIFERICHE		**2.94**	**4.28**	**0.38**	**0.55**	**0**	**0**	**A**

POTENZA TOTALE, CASO PEGGIORE (WATT)

	5V	12V		
Voltaggio	**5.25**	**9.45**	**12.60**	**TOT.**
Potenza	**22.47**	**5.20**	**0.00**	**27.67**

Tabella 17-1. Assorbimento di potenza del C128

CARICAMENTO DEL BUS

La Tabella 17-2 tratta in dettaglio il caricamento delle unità AC e DC per il sistema C128. Tutte le capacità sono in picofarad e le correnti sono in microampere.

Dispos.: Segnale:	DRAM	ROM	VIC	8563	4016	2332	LS TTL	MMU	PLA	8502	Z80	4066	SID CIA.	TOT. CAR.
TA_8-TA_{15}														
#	—	1	1	—	1	1	2	—	—	—	—	—	—	
I	—	10	2.5	—	10	10	800	—	—	—	—	—	—	833
C	—	8	8	—	4	8	30	—	—	—	—	—	—	58
SA_0-SA_7														
#	—	—	—	—	1	1	1	—	—	—	—	—	—	
I	—	—	—	—	10	10	400	—	—	—	—	—	—	420
C	—	—	—	—	4	8	15	—	—	—	—	—	—	27
D_0-D_7														
#	2	5	1	1	—	—	1	1	—	1	—	1	3	
I	20	50	2.5	2.5	—	—	400	2.5	—	2.5	—	0.1	7.5	488
C	—	40	8	8	—	—	15	8	—	8	—	8	24	129
R/W														
#	—	—	1	1	—	—	2	1	1	1	—	—	3	
I	—	—	2.5	2.5	—	—	800	2.5	2.5	2.5	—	—	7.5	820
C	—	—	8	8	—	—	30	8	8	8	—	—	24	94
RES														
#	—	—	—	1	—	—	—	1	—	1	1	—	3	
I	—	—	—	2.5	—	—	—	2.5	—	2.5	2.5	—	7.5	18
C	—	—	—	8	—	—	—	8	—	8	8	—	24	56
1MHz														
#	—	—	—	—	—	—	5	—	—	—	—	—	3	
I	—	—	—	—	—	—	2000	—	—	—	—	—	7.5	2008
C	—	—	—	—	—	—	75	—	—	—	—	—	24	99

Tabella 17-2. Requisiti di Potenza per il Caricamento del Bus (Capacità e Correnti)

SPECIFICHE DELL'AMBIENTE

Il C128 opera da 10 a 40 gradi Celsius (da 50 a 104 gradi Fahrenheit) e ad una umidità relativa non condensante da 5 a 95 %.

IL MICROPROCESSORE 8502

DESCRIZIONE GENERALE

L'8502 è un microprocessore di Tecnologia HMOSII, simile al 6510/6502. È il normale microprocessore operante usato in modo C64 ed in modo C128. È compatibile software con il 6510, perciò col 6502, l'8502 mette in funzione anche una porta della pagina zero usata nella gestione della memoria e nella implementazione della cassetta.

L'8502 viene anche specificato per il funzionamento a 2 MHz. Il funzionamento a 2 MHz è reso possibile rimuovendo il VIC dal sistema come un chip video. (Il chip VIC non viene mai rimosso completamente dal sistema C128, poichè continua a funzionare come generatore del clock e controllore di ricarica). Ci si riferisce al fatto che il VIC viene rimosso come chip video e coprocessore; così il ciclo completo del clock può essere dedicato al funzionamento del processore invece di condividere il ciclo con il VIC.

Il compito del processore di visualizzazione del video viene assunto dall'8563, che può funzionare senza avere bisogno di condividere il bus. Poichè le unità di I/O, SID, ecc., sono valutate solo ad 1 MHz, viene usata l'estensione del clock a 2 MHz per permettere a queste parti di essere raggiunte direttamente dal processore a 2 MHz e di mantenere ancora il rendimento funzionale al massimo.

Le unità di I/O non vengono toccate dal funzionamento a 2 MHz, poichè sono ancora pilotate da una fonte a 1 MHz (come tali, tutte le operazioni di timer rimangono invariate) e l'estensione del clock viene usata solo per sincronizzare il ciclo macchina a 2 MHz con il tempo alto a 1 MHz Φ0. Le fonti del clock e le capacità dell'estensione del clock sono generate dal chip VIC.

SPECIFICHE ELETTRICHE

Questa sezione descrive alcune limitazioni e specifiche elettriche del sistema.

GRANDEZZE MASSIME

La Tabella 17-3 dà i valori massimi assoluti del microprocessore 8502.

GRANDEZZA	SIMBOLO	VALORE	UNITÀ
Tensione di alimentazione	V_{cc}	-0.5 a + 7.0	Vdc
Voltaggio di Input	V_{in}	-0.5 a + 7.0	Vdc
Temperatura di memorizzazione	T_{stg}	-55 a + 150	°C
Temperatura di funzionamento	T_a	0 a + 70	°C

Tabella 17-3. Valori massimi assoluti dell'8502

CARATTERISTICHE ELETTRICHE

La Tabella 17-4 dà le specifiche elettriche base dell'8502 per le operazioni minima, tipica e caso peggiore, valide oltre l'intervallo di funzionamento T.

CARATTERISTICA	SIMBOLO	MIN	TIPO	MAX	UNITÀ
Voltaggio Alto di Input	V_{IH}				
Φ_0 (in)		$V_{ss}+2.4$	-	V_{cc}	Vdc
/RES, P_0-P_7,/IRQ, Dato		$V_{ss}+2.4$	-	-	Vdc
Voltaggio Basso di Input	V_{IL}				
Φ_0 (in)		$V_{ss}-0.3$	-	$V_{ss}+0.5$	Vdc
/RES, P_0-P_7, /IRQ, Dato		-	-	$V_{ss}+0.8$	Vdc
Corrente di dispersione di Input	I_{IN}				
(V_{in}=da 0 a 5.25V, V_{cc} =5.25V)					
Logica		-	-	2.5	μA
Φ_0 (in)		-	-	10.0	μA
Corr.3-Stati in Input (Off)	I_{TSI}				
(V_{in}=da 0.4 a 2.4V,V_{cc}= 5.25V)					
Linee dei Dati		-	-	10.0	μA
Voltaggio Alto di Output	V_{OH}				
(I_{OH} =-100μAdc,V_{cc}=4.75V)					
Dato, A_0-A_{15}, R/W, P_0-P_7		$V_{ss}+2.4$	-	-	Vdc
Voltaggio Basso di Output					
(I_{OL} =1.6mAdc,V_{cc}=4.75V)					
Dato, A_0-A_{15}, R/W, P_0-P_7		-	-	$V_{ss}+0.4$	Vdc
Corrente di Alimentazione	I_{cc}	-	125	-	mA
Capacità					
(V_{in}=0, T_a=25 C,f=1MHz)					
Logica,P_0-P_7	C_{in}	-	-	10	pF
Dato	C_{out}	-	-	15	pF
A_0-A_7	C_{out}	-	-	12	pF
Φ0	CΦ0	-	30	50	pF

Tabella 17-4. Specifiche elettriche di base dell'8502

DESCRIZIONE DEI SEGNALI

Ecco una descrizione di tutti i segnali dell'8502 da un punto di vista funzionale ed elettrico:

CLOCK (Φ_0) – Questo è il clock del sistema a velocità duale. Notate che il valore di input richiesto è oltre il caso peggiore TTL; così, si devono prendere ulteriori precauzioni se si tenta di inviare questo input da un livello di input standard TTL.

BUS DI INDIRIZZO (A_0 – A_{15}) – Output TTL. È in grado di inviare 2 caricamenti TTL in pF 130.

BUS DEI DATI (D_0 – D_7) – Bus a due direzioni per il trasferimento dei dati a/e dall'unità e le unità periferiche. Gli output sono buffer tri–stati in grado di inviare 2 caricamenti standard TTL in pF 130.

RESET – Questo input è usato per resettare o far partire il processore da una condizione di disinserimento.
Durante il periodo in cui questa linea è bassa, non si può scrivere in o dal processore. Se viene rilevato un segnale positivo dell'input, il processore inizierà immediatamente la sequenza di reset. Dopo un tempo di 6 cicli di inizializzazione del sistema, sarà posta la maschera di flag di interruzione di sistema ed il processore caricherà il contatore del programma dai contenuti delle locazioni della memoria $FFFC e $FFFD. Questa è la locazione di partenza per il controllo del programma. Dopo che V_{cc} ha raggiunto 4.75 volt in una routine di inserimento, si deve tenere il reset basso per almeno 2 cicli. A questo punto la linea R/W sarà valida.

RICHIESTA DI INTERRUZIONE (IRQ) – Input TTL; richede che il processore inizi una sequenza di interruzione. Il processore completerà l'esecuzione dell'istruzione corrente prima di riconoscere la richiesta. A quel punto, sarà esaminata la maschera di interruzione nel Registro del Codice di Stato. Se la maschera di interruzione non è posta, il processore inizierà una sequenza interruzione. Il Contatore del Programma ed il Registro di Stato del Processore saranno memorizzati nello stack e viene posto il flag di disabilitazione dell'interruzione così che non possano verificarsi altre interruzioni. Poi il processore caricherà il contatore del programma dalle locazioni di memoria $FFFE e $FFFF.

RICHIESTA DI INTERRUZIONE NON MASCHERABILE (NMI) – Input TTL, richiesta sensibile alla variazione negativa che il processore inizi una sequenza di interruzione. Il processore completerà l'esecuzione dell'istruzione corrente prima di riconoscere la richiesta. Il Contatore del Programma ed il Registro di Stato del Processore saranno memorizzati nello stack. Il processore allora caricherà il contatore del programma dalle locazioni di memoria $FFFA e $FFFB. Poichè NMI non è mascherabile, dovete fare attenzione ad assicurarvi che la richiesta NMI non dia luogo ad un incidente del sistema.

CONTROLLO DI ABILITAZIONE DELL'INDIRIZZO (AEC) – Il Bus di Indirizzo è valido solo quando la linea AEC è alta. Se è bassa, il bus di indirizzo è in uno stato di alta impedenza. Questo permette i DMA per i sistemi con processore duale.

PORTA DI I/O (P_0 – P_6) – Porta a due direzioni usata per trasferire i dati a/e dal processore direttamente. Il Registro Dati è localizzato in locazione $0001 ed il Registro di Direzione dei Dati è localizzato in locazione $0000.

R/W – Valore di output TTL dal processore per controllare la direzione del trasferimento dati tra il processore e la memoria, le unità periferiche, ecc. Questa linea è alta per la lettura della memoria e bassa per la scrittura.

RDY – Pronto. Valore di input TTL, usato per il DMA dell'8502. Il processore opera normalmente mentre RDY è alto. Quando RDY fa una transizione nello stato basso, il processore finirà l'operazione che sta compiendo e qualsiasi operazione seguente se è un ciclo di scrittura. Al verificarsi di un ciclo successivo di lettura il processore si fermerà, rendendo possibile accedere liberamente al bus del sistema.

TEMPORIZZAZIONE DEL PROCESSORE

Questa sezione analizza le considerazioni sulla temporizzazione dell'unità del processore dell'8502. La Tabella 17-5 è un grafo della temporizzazione del processore. La Figura 17-4 presenta i diagrammi della temporizzazione che mostrano sia la temporizzazione generale che il metodo particolare dell'estensione del clock usato nel modo a 2 MHz.

Caratteristiche Elettriche V_{cc} = 5v ± 5°/$_{0x}$, V_{ss} = Ov, Ts = da 0° C a 70° C

CARATTERISTICHE	SIMBOLO	MIN	MAX	UNITÀ
Tempo di predisposizione AEC	T_{AEC}	25	60	ns
Predisposiz. Dati Su Da Φ_0	T_{MDS}		100	ns
Tenuta Scrittura Dati Su	T_{HW}	40		ns
Bus Dati da rendere a tri.stati da AEC	T_{AEDT}		120	ns
Lettura Dati Stabile	T_{DSU}	40		ns
Tenuta Lettura dei Dati	T_{HR}	40		ns
Predisposiz. Indirizzo da Φ_0	T_{ADS}	40	75	ns
Tenuta Indirizzo	T_{HA}	40		ns
Predisposiz. Indirizzo da AEC	T_{AADS}		60	ns
Indirizzo tri-stati da AEC	T_{AEAT}		120	ns
Predisposiz. Porta di Input	T_{PDSU}	105		ns
Metà Porta Input	T_{PDH}	65		ns
Porta di Output Dati Valida	T_{PDW}		195	ns
Tempo del Ciclo	T_{CYC}	489		ns
Φ_0 (in) ampiezza dell'impulso @1.5V (clock di cristallo)	$P_{WH}\Phi 0$	235	265	ns
Φ_0 (in) tempo di salita	$TR\Phi 0$		10	ns
Φ_0 (in) tempo di caduta	$TF\Phi 0$		10	ns
Tempo di predisposizione RDY	T_{RDY}	80		ns

Tabella 17-5. Carta di temporizzazione del processore

ESTENSIONE DEL CLOCK

Quando si esegue in modo a 2 MHz, a volte il clock del processore deve essere esteso. Questo è gestito dal chip VIC, il processore ed il PLA operano insieme. Quando un'operazione di I/O viene decodificata durante un ciclo a 2 MHz, deve essere considerata la fase di relazione tra i clock a 2 MHz ed a 1 MHz. Se si verifica l'accesso a 2 MHz durante 1 MHz Φ1, l'accesso ad un chip di I/O tempificato sarà fuori sincronizzazione mentre il clock a 1 MHz piloterà tutti i chip di I/O. Così, durante la fase di relazione, /IOACC dal PLA segnala al chip VIC di estendere il clock a 2 MHz. Se dovessero verificarsi cicli a 2 MHz durante il ciclo a 1 MHz Φ2, non è necessaria una particolare attenzione.

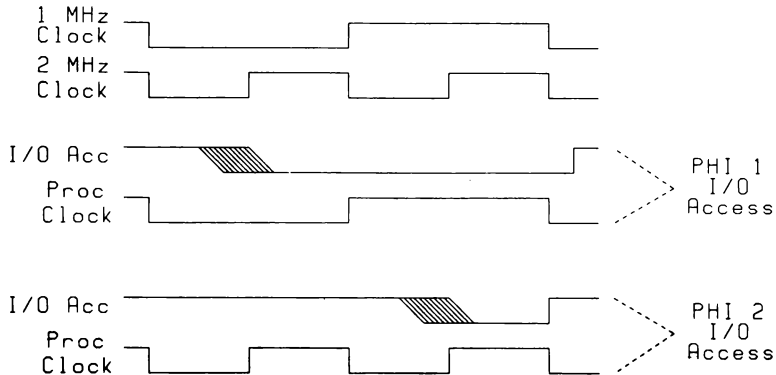



Figura 17-4. Estensione del Clock in modo 2 MHz

Notate le implicazioni della velocità. In modo a 2 MHz, si verificheranno metà degli accessi di I/O specificati ad una velocità effettiva di 1 MHz.

La Figura 17-5 è un diagramma della configurazione dei pin del microprocessore 8502.

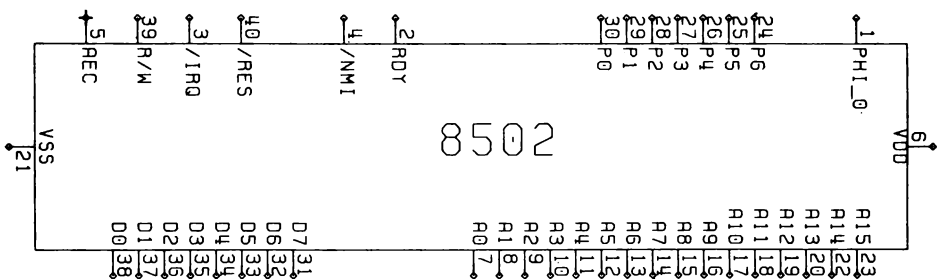



Figura 17-5. Configurazione dei pin del Microprocessore 8502

Per informazioni sulla programmazione in linguaggio macchina dell'8502 consultate il Volume 2, Capitolo 6, Linguaggio Macchina sul C128.

SPECIFICHE HARDWARE DEL MICROPROCESSORE Z80

Il microprocessore Z80 viene usato come un processore secondario nel C128 per eseguire i programmi a base CP/M. In questa sezione vengono trattate non solo l'operazione dello Z80 come parte del sistema C128, ma anche alcune importanti specifiche elettriche e di temporizzazione dello Z80.

DESCRIZIONE DEL SISTEMA

Lo Z80A, una versione a 4 MHz del processore standard Z80 dello Zilog, viene incluso come un processore alternativo del sistema C128. Questo permette al C128 di lanciare il sistema operativo CPM 3.0 ad una velocità effettiva di 2 MHz. Lo Z80 è interfacciato con l'interfaccia del bus 8502 e può accedere a tutte le unità cui può accedere lo Z80. L'interfaccia del bus dello Z80 (la parte più complessa dell'implementazione dello Z80) viene descritta in questa sezione, insieme al funzionamento dello Z80 come coprocessore del sistema C128.

> **NOTA:** Consultate la sezione sulla Descrizione dei Segnali più avanti in questo capitolo per le definizioni dei segnali menzionati nei paragrafi seguenti.

INTERFACCIA DEL BUS

Poichè un ciclo del bus dello Z80 è molto diverso da un ciclo del bus della famiglia 65xx, è richiesta una certa quantità di interfacciamento perchè uno Z80 controlli un bus di tipo 65xx. Poichè lo Z80 ha delle linee di controllo incorporate per l'arbitraggio del bus, è possibile isolare lo Z80 rendendo le linee di indirizzo tri-state. Così sia lo Z80 che l'8502 condividono linee di indirizzo comuni.

L'interfacciamento delle linee dei dati è più complesso. A causa della natura comune del bus in modo Z80, lo Z80 deve essere isolato dal bus mentre AEC è basso. Così, un buffer che può essere reso a tre stati deve pilotare il bus del processore durante le scritture dei dati Z80. Si verifica la situazione opposta durante una lettura Z80: lo Z80 non deve leggere cose che stanno procedendo mentre AEC è basso; deve trattenere (latch) il dato che era presente mentre AEC era alto. Così, un latch trasparente invia l'immissione dei dati allo Z80. Esso viene fermato dall'output abilita-lettura dello Z80 e trattenuto quando il clock da 1 MHz è basso. Si vedrà che lo Z80 gira realmente mentre AEC è basso, ma che il bus dei dati interfaccia con esso solo mentre AEC è alto.

INTERFACCIA DI CONTROLLO

L'interfacciamento di controllo dell'abilitazione della lettura dello Z80 deve fornire impulsi utili del clock allo Z80 e deve adattare i segnali Z80 ed abilita scrittura per il protocollo del bus di tipo 8502. Il clock Z80 viene fornito dal chip VIC ed è fondamentalmente un clock a 4 MHz che si verifica solo mentre AEC è basso, come mostrato nella Figura 17-6. Ciò assicura che lo Z80 venga tempificato solo quando agisce attivamente sul bus. Un'altra considerazione sulla temporizzazione dello Z80 è che mentre tutti i livelli dell'8502 e la maggior parte di quelli dello Z80 sono TTL compatibili, l'input del clock Z80 vuole livelli molto vicini a 5 volt. Per questo motivo, l'output del chip VIC viene elaborato dall'alimentatore a 9 volt; così, il circuito a 9 volts deve essere operazionale per lo Z80 ed il sistema in modo che funzionino. L'errore più comune di avviamento per il C128 è un salto della valvola di 9 volt.

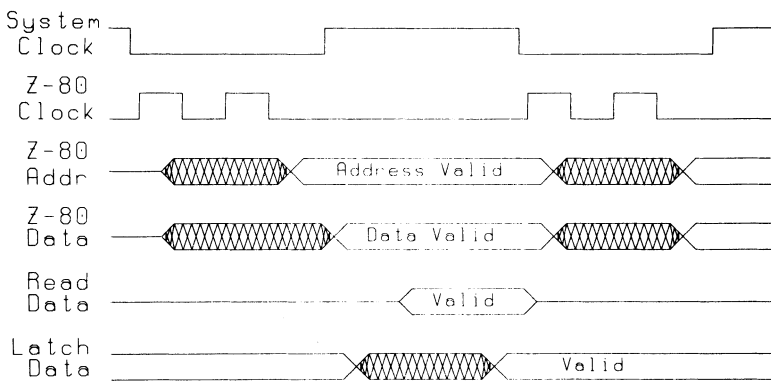



Figura 17-6. Temporizzazione del Bus Z80

COMMUTAZIONE DEL PROCESSORE

È importante per lo Z80 e l'8502 operare in un normale funzionamento come coprocessori, comunicando tra loro. Poichè solo un processore per volta può avere il bus, questo è solo un processo collettivo seriale, non parallelo o multiprocessing. Ciò è importante per molti motivi.

Primo, il sistema C128 deve accendersi con lo Z80 come processore principale. Questo per evitare che lo Z80 acceda accidentalmente al bus mentre sta accendendosi. Così, lo Z80 diventa supervisore dell'accensione e può fare ciò che vuole al bus. Poichè lo Z80 può iniziare certe applicazioni del C64 che causerebbero il guasto dell'8502, lo Z80 è la scelta logica come processore di inizio. Dopo alcune inizializzazioni, lo Z80 inizia l'8502 sia in modo C128 che in modo C64, dipendendo dalla presenza o meno di una cartuccia e dal tipo di cartuccia, se è presente. Il sistema operativo permette al modo C64 di essere forzato al momento dell'accensione.

Secondo, la commutazione del sistema permette allo Z80 di accedere alle routine Kernal dell'8502. Per i programmi standardizzati o per qualsiasi operazione di I/O non sopportata dal BIOS Z80, lo Z80 può passare il compito dell'I/O all'8502. Poichè lo Z80 vede la ROM BIOS dove l'8502 vede le sue pagine da 0 a

F, lo Z80 può operare senza paura di intralciare nessun puntatore 8502 o lo stack nel Banco 0 della RAM. Il BIOS ROM dello Z80 supera fisicamente quella sezione critica del Banco 1 della RAM.

Lo Z80 può ricevere un'assegnazione di richiesta del bus dall'MMU via /Z80EN, o dal chip VIC via BA. Poichè la linea di controllo del VIC viene usata per i DMA, l'ultima richiesta non è di interesse immediato. L'azione dello /Z80EN, tuttavia, è importante poichè è il meccanismo con il quale i due processori scambiano il controllo.

Quando la linea /Z80EN diventa alta, essa attiva un /BUSRQ Z80. Allora lo Z80 abbandona il bus portando /BUSACK basso. Quest'azione porta l'AEC dell'8502 alto e (facendo in modo che il VIC non richieda un DMA) porta la linea RDY dell'8502 alta, abilitando l'8502. Per la ricommutazione, un basso sul /BUSRQ dello Z80 risulterà nel portare /BUSACK Z80 alto, in modo che renda a tre stati l'8502 e che lo fermi. Consultate nell'ottavo volume l'Appendice K sul CP/M per dettagli sulla comunicazione tra chip.

DESCRIZIONE DEI SEGNALI

La lista che segue definisce ogni segnale dello Z80. La configurazione dei pin dello Z80 è mostrata nella Figura 17-7.

Bus di Indirizzo ($A_0 - A_{15}$): bus di indirizzo a 16 bit tri–stati. Usato per cicli di I/O a 16 bit. Permette fino a 256 porte di input e 256 porte di output. Al momento della ricarica, i 7 bit più bassi contengono un indirizzo di ricarica valido. (Questo segnale non è usato nel sistema C128).

Bus dei Dati ($D_0 - D_7$): bus di input/output in grado di rendere la ripartizione in tre stati; usato per scambi a 8 bit tra memoria e unità di I/O.

Ciclo Macchina Uno (/M_1): output, attivo basso. Questo segnale indica che il ciclo macchina corrente è prelievo del codice operativo di un'esecuzione di un'istruzione. Durante l'esecuzione di un codice operativo a due byte, viene generato /M_1, mentre viene preso ciascun byte. /M_1 si verifica anche con una richiesta di input/output (/IORQ) per indicare un ciclo di interruzione accettata. La linea /M_1 viene usata per disabilitare il decodificatore di I/O in un ciclo di interruzione accettata (consultare la Richiesta di I/O).

Richiesta di Memoria (/MREQ): basso attivo, output tri–stati che indica che il bus di indirizzo tiene un indirizzo valido per un'operazione di lettura o scrittura della memoria.

Richiesta di Input/Output (/INRQ): basso attivo, output tre–stati. Il segnale /INRQ indica che la metà inferiore del bus di indirizzo tiene un indirizzo valido per una operazione di I/O di lettura o scrittura. Un segnale /INRQ è generato anche da un segnale /M_1 quando viene accettata una interruzione per indicare che un vettore di interruzione di risposta può essere posto nel bus dei dati. Una interruzione può accettare durante /M_1; le operazioni di I/O non avvengono mai durante /M_1.

Lettura della Memoria (/RD): basso attivo, output tre-stati. /RD indica che il CPU vuole leggere i dati dalla memoria o da un'unità di I/O. Questo segnale viene usato generalmente per leggere-chiudere i dati nel bus dei dati.

Scrittura della Memoria (/WR): output attivo tre-stati basso. /WR indica che il bus dei dati tiene dati validi che possono essere elaborati dalla memoria o dalle unità di I/O.

Ricarica (/RFSH): output attivo basso usato per indicare che il bus di indirizzo tiene un indirizzo di ricarica nei suoi 7 bit più bassi. Così, il segnale corrente /MREQ dovrebbe essere usato per eseguire una ricarica della lettura di tutte le memorie dinamiche non ricaricate da una fonte alternativa. A7 viene posto a 0 e gli 8 bit più alti contengono il registro I in questo momento.

Stato di Arresto (/HALT): output attivo basso, che indica che lo Z80 ha eseguito un'istruzione di arresto e sta aspettando un tipo di interruzione prima che l'esecuzione possa continuare. Mentre è nello stato di arresto, lo Z80 esegue in continuazione NOP per continuare l'attività di ricaricamento.

Attesa (/WAIT): input attivo basso, usato per pilotare lo Z80 nello stato di attesa. Per tutto il tempo in cui questo segnale è basso, lo Z80 esegue lo stato di attesa; così, questo segnale può essere usato per accedere alla memoria lenta ed alle unità di I/O.

NOTA: Mentre lo Z80 è sia in uno stato di attesa che in uno stato di accettazione del bus (/BUSAK), non può essere eseguito un ricaricamento della memoria dinamica. Consultate l'Accettazione del Bus.

Richiesta di Interruzione (/INT): input attivo basso, pilotato da unità esterne. Se il flag interruzione IFF è abilitato e la linea di richiesta del bus (/BUSRQ) non è attiva, il processore riconosce la richiesta di interruzione alla fine dell'istruzione corrente. Quando lo Z80 riconosce un'interruzione, esso genera un segnale di riconoscimento dell'interruzione (/INRQ durante /M_1) all'inizio del ciclo successivo di istruzioni. Ci sono tre diversi modi di rispondere ad un'interruzione data. Consultate la Richiesta del Bus.

Interruzione Non Mascherabile (/NMI): input attivo basso. Questa interruzione è il segnale del trigger e non può essere mascherata di nuovo. Viene riconosciuta sempre alla fine dell'istruzione corrente, forzando lo Z80 a ricominciare dalla locazione $0066. Il contatore del programma viene salvato automaticamente nello stack per permettere un ritorno dal programma interrotto. Notate che cicli continui possono ritardare un /NMI impedendo la fine del ciclo corrente e che /BUSRQ escluderà /NMI.

Reset (/RESET): input basso attivo che forza il contatore del programma a zero ed inizializza lo Z80, che porrà il modo interruzione 0, disabiliterà le interruzioni e porrà i registri I e R a 0. Durante /RESET, i bus di indirizzo e dei dati divengono tri-stati e tutti gli altri segnali diverranno inattivi.

Richiesta Bus (/BUSRQ): input basso attivo che richiede che l'indirizzo del CPU, dei dati ed i segnali di controllo dell'output che possono essere tri-stati siano tri-stati per la condivisione e per DMA. Le linee divengono tri-state alla fine del corrente ciclo macchina.

Accettazione del Bus (/BUSAK): output basso attivo, usato per indicare a qualsiasi unità che assume il bus che lo Z80 è divenuto a tre stati e che il bus è stato concesso.

Clock (Φ): Sistema con clock a fase singola.

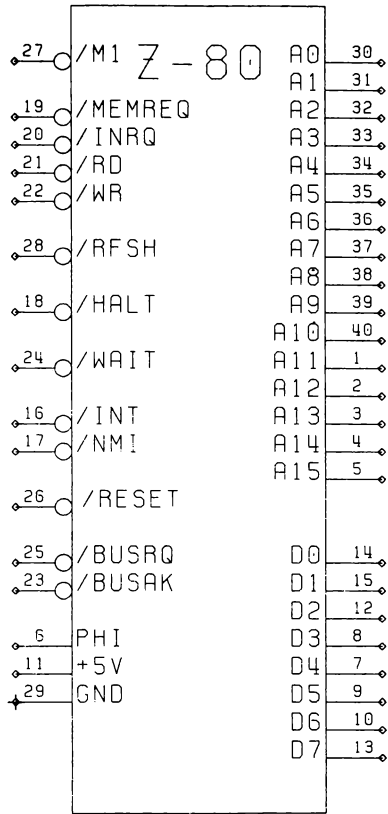



Figura 17-7. Configurazione dei pin del Microprocessore Z80

SPECIFICHE ELETTRICHE

VALORI MASSIMI ASSOLUTI

La Tabella 17-6 dà la temperatura massima assoluta, i valori di dissipazione di potenza e di voltaggio per lo Z80. Un danno permanente può verificarsi con grande probabilità se questi valori sono in eccesso.

PARAMETRO	SIMBOLO	INTERVALLO	UNITÀ
Temperatura di funzionamento	T_a	**da 0 a +70**	**° C**
Temperatura di memorizz.	T_{st}	**da -65 a +150**	**° C**
Voltaggio di input	V_{in}	**da -0.3 a 7.0**	**Vdc**
Dissipazione di potenza	P_{cc}	**1.5**	**W**

Tabella 17-6. Valori massimi assoluti dello Z80

CARATTERISTICHE OPERATIVE DC

La Tabella 17-7 mostra i valori operativi DC massimi per lo Z80. Tranne per quanto notato, questi valori sono applicabili per l'intero intervallo nominale della temperatura e del voltaggio.

PARAMETRO	SIMBOLO	INTERVALLO	UNITÀ
Variazione di tensione di alimentazione	V_{cc}	5 ± 5%	Vdc
Voltaggio basso di Input del Clock	V_{ILC}	da -0.3 a 0.8	Vdc
Voltaggio alto di Input del Clock	V_{IHC}	V_{cc} -0.6 a V_{cc} +0.3	Vdc
Voltaggio basso di Input	V_{IL}	da -0.3 a 0.8	Vdc
Voltaggio alto di Input	V_{IH}	da 2.0 a V_{cc}	Vdc
Voltaggio basso di Output (I_{OL} = 1.8mA)	V_{OL}	da V_{ss} a 0.4	Vdc
Voltaggio alto di Output (I_{OH} = - 250μA)	V_{OH}	da 2.4 a V_{cc}	Vdc
Corrente di Alimentaz. Potenza	I_{cc}	200	mA
Corrente di dispersione Input (V_{in} = da 0 a V_{cc})	I_{LI}	10	μA
Corrente di dispersione Tri-stati (V_{OUT}, V_{OH} V_{OUT} = V_{OL})	I_{LO}	+10,-10	μA
Corrente di dispersione per Input del Bus dei Dati ($V_{ss}<V_{in}<V_{cc}$)	I_{LD}	±10	μA

Tabella 17-7. Caratteristiche operative DC dello Z80

CAPACITÀ

I valori della linea di capacità per lo Z80 vengono dati nella Tabella 17-8. Tutte le misurazioni sono in T=25 gradi, F=1 MHz.

PARAMETRO	SIMBOLO	MASSIMO	UNITÀ
Capacità Clock	$C\Phi$	35	pF
Capacità di Input	C_{in}	5	pF
Capacità di Output	C_{out}	10	pF

Tabella 17-8. Valori della capacità

LA MATRICE LOGICA PROGRAMMATA (PLA)

La PLA 8721 del C128 è una versione programmata della Matrice Logica Pin Programmabile del Commodore 48 (Commodore Parte #315011). Essa fornisce tutte le selezioni del chip ed altri segnali decodificati che sono necessari per il C64, insieme ad un numero di tali segnali nuovi nel sistema C128. La Figura 17-8 mostra il chip della PLA.

La PLA fa una serie di cose che sono di vitale importanza per le operazioni del C128, comprese:

- Tutte le selezioni ROM (Kernal, BASIC, delle funzioni, esterne) in tutti i modi · operativi.
- La selezione del chip VIC.
- La selezione del colore del chip della RAM colore.
- La selezione del chip della RAM carattere.
- Abilitazione della scrittura nella RAM colore.
- Abilitazione della scrittura trattenuta in DRAM.
- Decodificazione della selezione dello Z80.
- Decodificazione dell'I/O dello Z80, per il ciclo di I/O dello Z80 e della formazione della mappa della memoria.
- Segnale di direzione del bus dei dati.
- Selezione del gruppo dei chip di I/O (incluso I/O-1, I/O-2, CIA-1, CIA-2, SID, 8563).
- Segnale di accesso dell'I/O che indica che si sta verificando un'operazione di I/O.
- Abilitazione CAS per l'abilitazione DRAM.

GENERAZIONE DELLA SELEZIONE DEL CHIP

Questa unità PLA è responsabile della definizione delle regole della formazione dei banchi nella ROM e nella RAM che il sistema seguirà. Questo chip genera selezioni del chip per tutta la ROM e per il chip VIC. Genera un'abilitazione per qualsiasi altra unità di I/O della mappa e può abilitare o disabilitare i CAS basati su ciò che è abilitato. In modo C128, si prendono decisioni usando gli indirizzi del processore e le linee di stato dei quattro modi: ROMBANKLO, ROMBANKHI, I/O SELECT e C128/64. Lo schema della formazione dei banchi in modo C128 è semplice e diretto. In modo Z80, il meccanismo di selezione diventa ancora più semplice, grazie al ciclo di I/O del processore Z80.

Il chip C64 seleziona un conto per la massa della fonte PLA. Il C64 seleziona l'I/O, la RAM e la ROM basate sulle linee di controllo interne: BA, HIRAM, LORAM e CHAREN. Lo stato di queste linee e di indirizzi decodificati, determi-

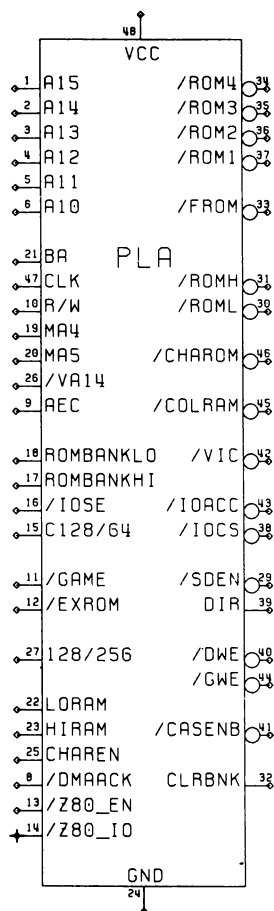



Figura 17-8. Chip PLA del C128

nano quale chip viene selezionato (se ce ne sono) ogni volta. Se viene inserita una cartuccia, entrano in gioco due ulteriori linee di controllo: /EXROM e /GAME. Varie combinazioni di queste linee causano l'affermazione di diverse mappe della memoria, tutte basate sulla fonte PLA.

ALTRE FUNZIONI

La PLA esegue una varietà di funzioni diverse dalle selezioni dei chip. Essa crea gli impulsi stroboscopici per l'abilitazione della scrittura sia per DRAM che per la RAM Colore. In modo C128, non si ha bisogno delle linee di controllo C64 (HIRAM, LORAM e CHAREN), poichè l'MMU controlla il metodo più sofisticato di formazione dei banchi del C128. Così, queste linee vengono usate per ampliare la funzionalità del C128 ad un costo minimo o non ulteriore per l'hardware. La linea CHAREN è usata in modo C128 per accendere e spegnere la ROM carattere nel banco VIC selezionato; in modo C128 la ROM può apparire o scomparire in qualsiasi banco VIC.

La seconda delle nuove funzioni usa LORAM e HIRAM per selezionare uno dei due banchi della RAM Colore. Il valore LORAM seleziona il banco che sarà visto durante il tempo del processore; il valore di HIRAM seleziona il banco che sarà visto durante il tempo del VIC. Così un programma può passare tra due figure a colori in modo molto netto, o il processore può modificare una figura a colori mentre ne visualizza un'altra.

UNITÀ DI GESTIONE DELLA MEMORIA (MMU)

L'MMU è progettata per avere un controllo complesso delle risorse del sistema della memoria del C128. Per il modo in cui gestisce tutti i modi standard delle operazioni del C64, è completamente compatibile col C64. Inoltre, essa controlla la gestione dei modi C128 particolari incluso il modo Z80. Le funzioni MMU includono:

- La generazione del bus di indirizzo tradotto (TA8-TA15).
  La generazione dei segnali di controllo per diversi modi del processore (C128, C64, Z80).
- La generazione di linee di selezione CAS per la formazione dei banchi della RAM.
- La generazione delle linee ROMBANK per la formazione dei banchi della ROM.

La programmazione dell'MMU è descritta in dettaglio nel quinto Volume, Capitolo 14.

DESCRIZIONE FISICA

Molti dei segnali di input ed output dell'MMU sono stati trattati fino ad ora in modo informale. Questa sezione contiene le descrizioni dell'MMU come unità fisica a 48 pin, compresa una descrizione di tutti i requisiti dei pin, dei segnali di input ed output ed i requisiti elettrici.

REQUISITI DEI PIN

La Tabella 17-9 lista i pin richiesti dall'MMU, indicando il numero per ogni categoria del segnale ed il numero complessivo.

NOMI SEGNALI	DESCRIZIONE	NUM DI PIN
A_0-A_3,A_8-A_{15}	Linee di Indir In	12
$A_{4/5}$,$A_{6/7}$	Linee di Indir Combinate In	2
D_0-D_7	Linee Dati In/Out	8
TA_8-TA_{15}	Linee di Indir Tradotto Out	8
V_{cc}	+5V	1
GND	Terra	1
PHI_0	2 MHz Φ Clock In	1
RESET	Reset del Sistema In	1
R/W	Linea Lettura/Scrittura In	1
/CAS_0-/CAS_1	DRAM CAS,64K del Banco Out	2
AEC	Controllo Abilitaz Indirizzo In	1
/Z80EN	Abilitazione Z80 Out	1
/GAME	Abilitaz Game ROM In, Controllo Out	1
/EXROM	Abilitaz ROM esterna In, Controllo Out	1
MS_0-MS_1	Stato della Memoria Out	2
I/O (MS_2)	Selezione I/O Out	1
C128/64 (MS_3)	Modo C128 o C64 Out	1
40/80	Stato 40/80 In	1
/FSDIR	Direzione Seriale Veloce Out	1
MUX	Memoria Multiplex In	1
	TOTALE	48

Tabella 17-9. Requisiti dei pin MMU

DESCRIZIONE DEI PIN

Ecco una descrizione dettagliata dei segnali MMU di input e di output. La Figura 17-9 mostra la configurazione dei pin dell'MMU. È qui inclusa qualsiasi opzione vincolata disponibile.

I segnali di input della MMU sono:

- A_0-A_3,A_8-A_{15}: Indirizzi dal microprocessore. Usati per derivare le selezioni dei chip e linee di indirizzo multiplex. A_0-A_3 si trovano ai pin 18-21 della MMU, mentre A_8-A_{15} sono localizzati ai pin 24-31.
- $A_{4/5}$,$A_{6/7}$: Indirizzi combinati dal microprocessore. Usati insieme agli indirizzi semplici, combinati in questa forma per abbassare il conto dei pin della MMU. Localizzati rispettivamente ai pin 22 e 23.
- PHI_0: Clock del presistema. Usato per una transizione precoce dei segnali bloccati (gated) nelle operazioni di scrittura. L'indirizzo del processore è valido nel fianco di salita ed il dato è valido nel fianco di caduta. Questo si trova al pin 33.
- R/W: Linea di controllo del Sistema di Lettura/Scrittura. Questo input è alto per una lettura del processore, basso per una scrittura del processore. Questo segnale è localizzato al pin 32 nel chip della MMU.
- RESET: Reset del Sistema. Questo input inizializza i registri interni al momento dell'accensione o di un reset dell'hardware. Si può trovare al pin 2.

- AEC: Controllo di Abilitazione dell'indirizzo. Indica se il processore 8502 o il VIC ha accesso al bus condiviso. Se è basso, il VIC o un DMA esterno ha il bus e VA16 hanno il bus del processore e non avviene alcuna traduzione del puntatore o BIOS. Il segnale occupa il pin 16.
- MUX: Segnale multiplex della memoria, usato per sincronizzare varie sezioni della MMU. È localizzato al pin 17.
- V_{dd}: Alimentazione del sistema + 5Vdc, connessa al pin 1.
- V_{ss}: Terra del sistema, connessa al pin 34.

Segue una rappresentazione delle linee bidirezionali della MMU. Alcuni dei bit delle porte trattati qui in dettaglio sono lasciati a future espansioni in senso monodirezionale.

- D_0-D_7: Input dati dal microprocessore. Usati per la scrittura nei registri interni. Localizzati nei pin da 35 a 42.
- /EXROM: Questo segnale viene usato per leggere la linea /EXROM del connettore di espansione in modo C64 e come linea di controllo dell'espansione in modo C128. È localizzato nel pin 46. Questa linea piloterà in output un caricamento TTL ed ha un'elevazione del modo di impoverimento passivo in input. Questo segnale può essere ridotto ma non elevato da un driver esterno.
- /GAME: Questo segnale è usato per leggere la linea /GAME nel connettore di espansione in modo C64 e come linea di controllo del banco della RAM colore in modo C128. È localizzato al pin 45. Questa linea piloterà in output un caricamento TTL ed ha un'elevazione del modo di impoverimento passivo in input. L'hardware esterno può ridurre questa linea, ma non la può elevare.
- 40/80: Questa porta in modo input legge il commutatore delle colonne 40/80. Rileva se questa commutazione è chiusa o no. La sua funzione di output è aperta all'espansione. È localizzata nel pin 48. Questa linea porterà in output un caricamento TTL ed ha un'elevazione del modo di impoverimento passivo in input. L'hardware esterno può ridurre questa linea ma non elevarla.
- FSDIR: Questa porta in modo output è usata per controllare la direzione dei dati dell'interfaccia seriale veloce del dischetto. È un segnale della porta generale ed è connesso al pin 44. Questa linea porterà in output un caricamento TTL ed ha un'elevazione del modo di impoverimento passivo in input. L'hardware esterno può ridurre/abbassare questa linea ma non la può elevare.

La lista seguente rappresenta i segnali di output della MMU, le loro locazioni fisiche nella MMU ed i loro livelli logici se sono applicabili.

- TA_8-TA_{15}: Output dell'indirizzo tradotto. A tre stati per i cicli VIC durante AEC, essi forniscono indirizzi fisici tradotti per l'uso nel Bus di Indirizzo Multiplex e nel Bus Fisso Condiviso. Da TA_{12} a TA_{15} sono definiti per avere un'elevazione interna del modo di impoverimento passivo con una resistenza equivalente di 3.3KΩ. Da TA_8 a TA_{11} ognuno diventa tri-stato durante

VIC (AEC basso). Questi sono localizzati nella MMU nei pin da 10 a 3.

- MS_0-MS_1: Chiamati anche ROMBANK0 e ROMBANK1, questi output controllano la formazione dei banchi della ROM per tutti gli slot della ROM. Essi sono localizzati nei pin 15 e 14. Queste linee vengono usate per decodificare la selezione dei banchi della ROM per qualsiasi accesso ROM in modo C128. Se sono entrambi bassi, è stato selezionato un sistema ROM. Se solo MS_1 è alto, allora è stata selezionata una funzione ROM incorporata. Se solo MS_0 è alto, allora è stata selezionata una funzione ROM esterna. Infine, se entrambi sono alti, è stata selezionata la RAM che occupa lo slot particolare. In modo C64, PLA ignora completamente queste linee.
- I/O: Questo output viene usato per selezionare gli I/O della mappa della memoria in modo C128. Esso è nel pin 13 ed è conosciuto anche come MS_2. In modo C128, questa linea riflette sempre la polarità del bit di I/O. Viene ignorato da PLA in modo C64 e rimane alto per tutto il modo C64.
- C128: Questo output porta il sistema ad agire sia in modo C128 che in modo C64. È localizzato nel pin 47 ed è conosciuto anche come MS3. Diventa basso per indicare il modo C64, alto per il modo C128.
- /Z80EN: Questo output viene usato per abilitare il processore Z80 e disabilitare la normale operazione del processore 8502. Si può trovare nel pin 43. Diventa basso per indicare il modo Z80, alto per tutti gli altri modi.
- /CAS_0-CAS_1: CAS abilita il controllo della formazione dei banchi della RAM. CAS_0 abilita il primo banco di 64K; CAS_1 abilita il secondo banco di 64K. Questi sono i pin 12 e 11 rispettivamente.

VALORI MASSIMI ASSOLUTI

ITEM	SIMBOLO	INTERVALLO	UNITÀ
Voltaggio di Input	V_{in}	**da -2.0 a +7.0**	**Vdc**
Voltaggio di Alimentaz.	V_{cc}	**da -2.0 a +7.0**	**Vdc**
Temperatura di Funzionam.	T_a	**da 0 a 70**	**°C**
Temperatura di Memorizz.	T_{st}	**da -55 a 150**	**°C**

Tabella 17-10. Valori massimi assoluti per l'MMU

CONDIZIONI MASSIME DI FUNZIONAMENTO

ITEM	SIMBOLO	INTERVALLO	UNITÀ
Variazione voltaggio	V_{cc}	5.0 + 5%	Vdc
Corrente di dispersione di input	I_I	-1.0	µA
Voltaggio alto di input	V_{IH}	da V_{ss}+2.4 a Vcc+1.0	Vdc
Voltaggio basso di input	V_{IL}	da V_{ss}-2.0 a Vss+0.8	Vdc
Voltaggio alto di output (I_{OH} = -200µ. V_{cc} = 5V ± 5%)	V_{OH}	V_{ss} + 2.4	Vdc
Voltaggio basso di output (I_{OL} = -3.2mA, V_{cc} = 5V ± 5%)	V_{OL}	V_{ss} + 0.4	Vdc
Corrente massima di input	I_{cc}	250	mA

Tabella 17-11. Condizioni massime di funzionamento per l'MMU

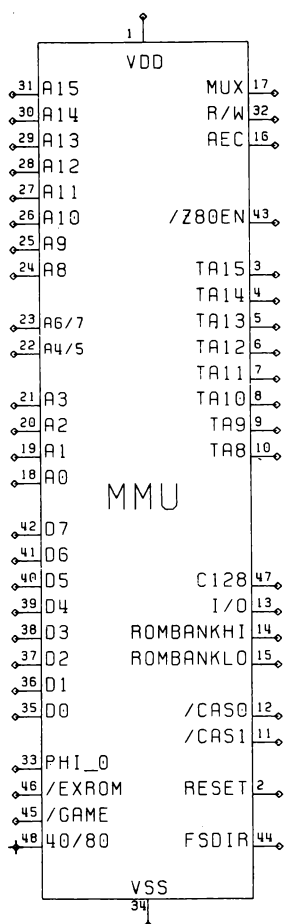



Figura 17-9. Configurazione dei pin dell'Unità di gestione della memoria

CHIP DELL'INTERFACCIA VIDEO 8564

Il chip VIC 8564 usato in C128 è una versione aggiornata del chip VIC usato nei sistemi correnti del C64. Esso contiene tutte le capacità video del precedente chip VIC 6567, inclusi la grafica bit-map ad alta risoluzione e i blocchi di immagini mobili. Esso fornisce anche nuove funzioni usate dal sistema C128, inclusa la scansione della tastiera estesa. La sua mappa dei registri è compatibile verso l'alto con il vecchio VIC, permettendo la compatibilità in modo C64. Viene acceso da una singola sorgente 5V DC invece delle due sorgenti richieste dal vecchio chip VIC. La configurazione dei pin dell'8564 è mostrata nella Figura 17-10.

DESCRIZIONE GENERALE

Il chip VIC 8564 è simile al chip VIC 6567, e fornisce molte funzioni nuove uniche per i requisiti del sistema C128. Esso girerà su un'alimentazione singola di 5V DC ed è impacchettato in un contenitore DIN (dual-in-line) a 48 terminali.

OPERAZIONI DEL VIC 8564

Il VIC 8564 sopporta tutte le operazioni del chip VIC precedente. Queste funzioni, riassunte in breve, sono:

- Modo visualizzazione del carattere colore standard.
- Modo visualizzazione del carattere multicolore.
- Modo visualizzazione del carattere colore esteso.
- Modo bit map standard.
- Modo bit map multicolore.
- Blocchi di immagini mobili.
- Ingrandimento del blocco di immagini mobili.
- Priorità del blocco di immagini mobili.
- Registrazione della collisione del blocco di immagini mobili.
- Azzeramento dello schermo.
- Selezione della visualizzazione riga/colonna.
- Scroll dolce.
- Penna ottica.
- Interruzione della comparazione del raster.

Poichè queste funzioni esistono nel VIC precedente, la loro descrizione qui è minima, mentre le informazioni per la programmazione del VIC sono nel Capitolo 9. Le nuove funzioni sono descritte nei dettagli di seguito.

SCANSIONE DELLA TASTIERA ESTESA

L'8564 contiene un registro chiamato il Registro di Controllo della Tastiera. Questo registro permette la scansione di tre linee di controllo ulteriori della tastiera riferite alla tastiera del C128. Così, la tastiera C128 può avere tasti ulteriori superiori in modo C128 mentre mantiene ancora la compatibilità con la tastiera C64 in modo C64. In questo registro (registro 53295 ($D02F)), i bit 0-2 sono riflessi direttamente in linee di output dalla K0 alla K2, mentre i bit 3-7 non sono usati, ritornando alti quando sono letti.

FUNZIONAMENTO A 2 MHz

Il chip VIC contiene un registro che permette al sistema C128 di funzionare a 2 MHz invece che alla velocità standard di 1 MHz del C64. Questa velocità di funzionamento, tuttavia, non permette di usare il chip VIC come processore di visualizzazione. Questo è il bit 0 in 53296 ($D030), e il porre questo bit abilita il modo a 2 MHz. Durante le operazioni a 2 MHz, il VIC è disabilitato come processore del video. Il processore passa l'intero tempo del ciclo sul bus, mentre il VIC è responsabile solo della ricarica dinamica della RAM. L'azzeramento di questo bit riporterà l'operazione ad 1 MHz e permetterà l'uso del VIC come chip di visualizzazione del video. Durante la ricarica e l'accesso di I/O, il clock del sistema è forzato ad 1 MHz senza tenere conto del posizionamento di questo bit.

La velocità a 2 MHz è disponibile in modo C64 posizionando il bit 0 della locazione 53296 ($D030). Prima di questo, azzerate lo schermo cancellando il bit 4 di locazione 53265 ($D011). Poi potete procedere a 2 MHz (per eseguire il crunch del numero, per esempio); tuttavia, non avrete lo schermo VIC visibile. Per ritornare, ponete il bit 4 di 53265 ($D011) ed azzerate il bit 0 di 53296 ($D030).

Il bit 1 di questo registro contiene una funzione di test del chip. Per un normale funzionamento, questo bit deve essere azzerato. Nessuno degli altri bit è connesso in questo registro.

CONTROLLO DEL CLOCK DEL SISTEMA

Il nuovo chip VIC genera parecchi clock usati dal sistema C128. Il clock principale è il clock ad 1 MHz, che opera approssimativamente a 1 MHz tutte le volte. La maggior parte delle operazioni del bus e tutte le operazioni di I/O avvengono in relazione a questo clock. Il clock successivo da considerare è il clock a 2 MHz. Questo clock temporizza le componenti selezionate dei sistemi, come il processore, a 2 MHz se si è il modo 2 MHz. Il chip VIC visualizza l'input /IOACC, che indica l'accesso di un chip di I/O e quando è sostenuto, esso forzerà il clock a 2 MHz per sincronizzare tutte le parti a 2 MHz con le parti di I/O a 1 MHz. Infine, l'ultimo clock è il clock Z80, che è un clock a 4 MHz che prende luogo solo durante la metà bassa del clock a 1 MHz. Poichè tutte le parti di I/O tempificate guardano solo il clock ad 1 MHz, tutti i calcoli dei tempi di I/O rimarranno uguali senza tenere conto di ciò che fa il clock a 2 MHz.

DESCRIZIONE DEI SEGNALI

Il chip VIC è montato su un contenitore dual-in-line a 48 terminali. La lista seguente descrive i segnali elettrici che esso genera.

I segnali usati nell'interfaccia del sistema sono:

- **D_0-D_7**: Questi sono i segnali bidirezionali del bus dei dati. Essi servono per la comunicazione tra il VIC ed il processore e possono essere raggiunti solo durante AEC alto. Occupano i pin da 7 a 1 e 47, rispettivamente.
- **D_8-D_{11}**: Questi sono i segnali del bus dei dati esteso. Essi sono usati per la comunicazione del VIC con la RAM colore. Occupano i pin da 47 a 43 in ordine.
- **/CS**: Selezione del chip, usato dal processore per selezionare il chip VIC. Si trova al pin 13.
- **R/W**: Lettura/scrittura standard del bus 8502 per l'interfacciamento tra il processore ed i vari registri VIC. Pin 14.
- **A_0-A_8**: Linee di indirizzo multiplex, pin da 32 a 38. Durante il tempo dell'indirizzo di riga, A_0-A_8 sono portati su A_0-A_5. Durante il tempo dell'indirizzo di colonna, A_8-A_{13} sono portati su A_0-A_5 ed A_6 è tenuto ad 1. Durante una lettura o scrittura del processore, A_0-A_5 operano come input dell'indirizzo che fanno da latch sul fianco basso o su /RAS.
- **A-A_{10}**: Linee di indirizzo fisso, pin da 39 a 42. Queste linee di indirizzo sono usate per accessi non multiplex alla memoria del VIC, come alla ROM carattere ed alla RAM colore.
- **1MHz**: Il clock del sistema ad 1 MHz, pin 18. L'intera attività del bus del sistema fa riferimento a questo clock.
- **2MHz**: Questo è il clock del sistema di cambio, che sarà sia a 1 MHz che a 2 MHz. Se il bit a 2 MHz è azzerato, non avviene nè VIC nè DMA esterno nè avviene alcuna operazione di I/O, il clock sarà di 2 MHz; altrimenti sarà 1 MHz. Si trova al pin 23.
- **Z80 Phi**: Il clock speciale Z80 a 4 MHz, pin 25.
- **/IOACC**: Da PLA, indica un accesso di I/O al chip per l'estensione del clock. Pin 22 del VIC.
- **/RAS**: Impulso stroboscopico dell'indirizzo di riga per i DRAM, pin 19.
- **/CAS**: Impulso stroboscopico per l'indirizzo di colonna per i DRAM, pin 20.
- **MUX**: Controllo dell'indirizzo multiplexing per i DRAM, pin 21.
- **/IRQ**: Output di interruzione, usato per segnalare che una delle varie sorgenti di interruzione interna ha avuto luogo. Richiede un innalzamento, si trova al pin 8 del chip.
- **AEC**: Controllo di Abilitazione dell'Indirizzo, alto per l'abilitazione del processore del bus condiviso, basso per il ciclo VIC e per il VIC o DMA esterno. Si trova al pin 12 del VIC.
- **BA**: Segnale di Bus Disponibile, usato per il processore DMA. Pin 10.
- **K_0-K_2**: Bit dell'impulso stroboscopico per la tastiera estesa, pin da 26 a 28.
- **LP**: Latch triggerato del segnale per l'input della penna ottica. Pin 9 del chip VIC.

I segnali compresi nell'interfaccia video, cioè tutti i segnali richiesti per creare un'immagine a colori, sono:

- PH IN: La velocità di spostamento fondamentale del clock, chiamata anche dot clock. Usata come riferimento per tutti i clock del sistema. Localizzata al pin 30.
- PH CL: Il clock colore, usato per derivare il segnale del colore (croma). Pin 29.
- SYNC: Output contenente informazioni sul sincronismo composto, informazioni sul video sui dati e sulla luminosità. Richiede un'elevazione, pin 17.
- COLOR: Output contenente tutte le informazioni su video a colori. Sorgente aperta in output, dovrebbe essere collegata a terra per mezzo di un resistore. Si trova al pin 16.

SPECIFICHE ELETTRICHE

Le Tabelle 17-12 e 17-13 specificano le operazioni elettriche del chip VIC nel suo nuovo formato. Questi specchietti includono i valori massimi assoluti e le condizioni di massimo funzionamento.

ITEM	SIMBOLO	INTERVALLO	UNITÀ
Voltaggio di Input	V_{in}	**da -0.5 a +7.0**	**Vdc**
Tensione di Linea	V_{cc}	**da -0.5 a +7.0**	**Vdc**
Temperatura di Funzion.	T_a	**da 0 a 75**	**°C**
Temperatura di Memorizz.	T_{st}	**da -65 a 150**	**°C**

Tabella 17-12. Valori massimi assoluti per il Chip VIC

Segue una lista delle specifiche massime di funzionamento per il nuovo chip VIC:

ITEM	SIMBOLO	INTERVALLO	UNITÀ
Variazione di tensione di Alimentaz.	V_{cc}	$5.0 \pm 5\%$	**Vdc**
Corrente di dispersione di Input	I_I	**-1.0**	**μA**
Voltaggio alto di Input	V_{IH}	da $V_{ss}+2.0$ a V_{cc}	**Vdc**
Voltaggio basso di Input	V_{IL}	da $V_{ss}-0.5$ a $V_{ss}+0.8$	**Vdc**
Voltaggio alto di Output ($I_{OH}=-200\mu A$, $V_{cc}=5.0 \pm 5\%$ Vdc)	V_{OH}	$V_{ss}+2.4$	**Vdc**
Voltaggio basso di Output ($I_{OL}=-3.2mA$, $V_{cc}=5.0+5\%$ Vdc)	V_{OL}	$V_{ss}+0.4$	**Vdc**
Corrente massima di Alimentaz.	I_{cc}	**200**	**mA**

Tabella 17-13. Massime specifiche di funzionamento per il Chip VIC

REGISTRO DEL RASTER

Il **Registro del Raster** è un registro con funzione duale. Una lettura del registro del raster 53266 ($D012) ritorna gli 8 bit più bassi della posizione corrente del raster [MSB-RC8 è localizzato nel registro 53265 ($D011)]. Una scrittura nei bit del raster (RC8 incluso) è trattenuta per l'uso in una comparazione interna del raster. Quando il raster corrente corrisponde al valore scritto, viene posto il latch di interruzione del raster. Il registro del raster dovrebbe essere interrogato per prevenire il farfallio del video ritardando i cambiamenti del video perchè si verifichino fuori dell'area visibile. Quest'area visibile dello schermo va dal raster 51 al raster 251 ($033-$0FB).

REGISTRO DI INTERRUZIONE

Il **Registro di Interruzione** indica lo stato delle quattro sorgenti di interruzione. Un latch di interruzione nel registro 53273 ($D019) è posto a 1 quando una sorgente di interruzione ha generato una richiesta di interruzione.

BIT DI LATCH	BIT DI ABILITAZ.	SE POSTO
IRST	ERST	conto reale del raster = conto del raster memorizzato (bit 0)
IMDC	EMDC	collisione MOB-DATA (solo primo bit) (bit 1)
IMMC	EMMC	collisione MOB-MOB (solo primo bit) (bit 2)
ILP	ELP	prima transizione negativa di LP per frame (bit 3)
IRQ		se IRQ/output basso (bit 7)

Tabella 17-14. Definizioni del Registro di interruzione

Per abilitare una richiesta di interruzione in modo che ponga l'output IRQ/ a 0, il bit corrispondente abilitatore dell'interruzione nel registro 53274 ($D01A) deve essere posto a "1". Una volta che un latch interruzione è stato posto, il latch può essere azzerato solo scrivendo un "1" nel bit associato del registro interruzione. Questa funzione permette un trattamento selettivo delle interruzioni video senza che il software memorizzi le interruzioni attive.

AZZERAMENTO (BLANKING) DELLO SCHERMO

Lo schermo visualizzato può essere azzerato riportando il bit **BLNK** (bit 4) del registro 53265 ($D011) a zero (0). Quando lo schermo è azzerato, l'intero schermo visualizza il colore esterno specificato dal registro 53280 ($D020). Quando l'azzeramento è abilitato, sono richiesti solo gli accessi trasparenti alla memoria (fase 1), permettendo l'utilizzazione completa da parte del processore del bus del sistema. Tuttavia, i dati degli sprite saranno reggiunti se gli sprite non sono disabilitati.

SELEZIONE DELLA RIGA/COLONNA DELLO SCHERMO

Il normale schermo visualizzato consiste di 25 righe di 40 caratteri. Per scopi di visualizzazione speciali, la finestra dello schermo è ridotta a 24 righe di 38 caratteri. Non c'è modifica del formato dell'informazione del display, tranne che i caratteri (bit) adiacenti all'area del margine esterno sono coperti dal margine stesso.

RSEL	NUMERO DELLE RIGHE	CSEL	NUMERO DELLE COLONNE
0	**24 righe**	**0**	**38 colonne**
1	**25 righe**	**1**	**40 colonne**

Il bit **RSEL** (bit 3) è nel registro 53265 ($D011) ed il bit CSEL (bit 3) è nel registro 53270 ($D016). Per una visualizzazione standard, viene normalmente usata la finestra di visualizzazione più grande, mentre la finestra di visualizzazione più piccola viene normalmente usata insieme allo scrolling.

SCROLLING

I dati della visualizzazione possono essere spostati in alto di un carattere sia in direzione orizzontale che verticale. Con la finestra di visualizzazione più piccola (in alto), lo scrolling può essere usato per creare un movimento dolce di panoramica dei dati visualizzati mentre si aggiorna la memoria del sistema solo quando è richiesta una nuova riga (o colonna) di caratteri. Lo scrolling viene usato anche per centrare una visualizzazione dentro i margini. Un esempio di scrolling dolce orizzontale si trova nel Capitolo 9.

BIT	REGISTRO	FUNZIONE
0-2	53270 ($D016)	Posizione orizzontale
0-2	53265 ($D011)	Posizione verticale

PENNA OTTICA

L'input della penna ottica agisce come latch per la posizione corrente dello schermo in un paio di registri (LPX, LPY) di un segnale lento (low-going edge). Poichè la posizione X è definita da un contatore di 9 bit (53267 ($D013)), viene fornita una risoluzione a due punti orizzontali. In modo simile, la posizione Y subisce un latch nel registro 53268 ($D014) con 8 bit che danno una risoluzione unica del raster all'interno del video visibile. Il latch della penna ottica può essere triggerato solo una volta per frame e trigger successivi all'interno della stessa frame non avranno effetto.

Per ulteriori informazioni sulla programmazione del chip VIC (8564), consultate il Capitolo 9, La Potenza nascosta della Grafica del Commodore 128.

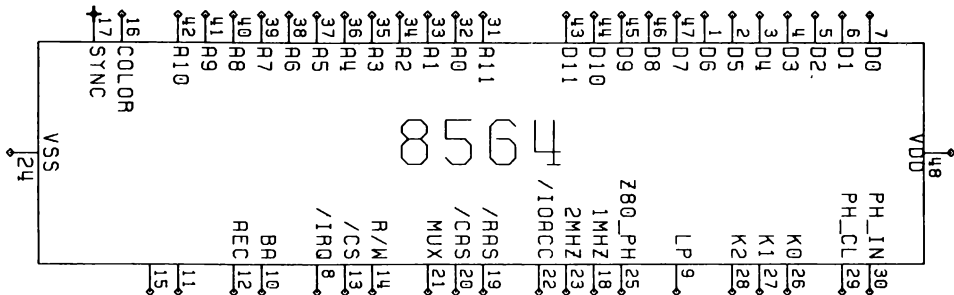



Figura 17-10. Chip VIC 8564

L'UNITÀ DI CONTROLLO DEL VIDEO 8563

L'8563 è un'unità di controllo di visualizzazione del video a colori della tecnologia HMOSII custom ad 80 colonne. L'8563 fornisce tutti i segnali necessari per l'interfacciamento diretto con i 16K di DRAM, compresa la ricarica e l'RGBI generato per l'uso di un monitor esterno RGBI. Per ulteriori informazioni sull'unità di controllo del video 8563, consultate il Capitolo 9, la Programmazione del Chip ad 80 Colonne (8563).

DESCRIZIONE GENERALE

L'8563 è un chip di visualizzazione del testo progettato per implementare un sistema di visualizzazione ad 80 colonne con il minimo di elementi e di costi. Il chip contiene la logica di frequenza con pixel ad alta velocità necessaria per i video RGBI ad 80 colonne. Può pilotare i caricamenti direttamente, sebbene sia preferibile la bufferizzazione nella maggior parte delle applicazioni del mondo reale. Il chip può indirizzare fino a 64K di DRAM per i caratteri, puntatori dei caratteri ed informazioni sugli attributi. Il chip fornisce RAS, CAS, l'abilitazione della scrittura, l'indirizzo, dati e ricarica per le sue DRAM subordinate. Un bit programmabile seleziona sia due DRAM 4416 (16K in totale) o otto DRAM 4164 (64K in totale) per la RAM video. Il sistema C128 usa le DRAM 4416.

REGISTRI ESTERNI

L'8563, che risiede in $D600 nel C128, appare all'utente come un'unità consistente di solo due registri. Questi due registri sono registri indiretti che devono essere programmati per accedere all'insieme interno dei 37 registri di programmazione. Il primo registro, localizzato in $D600, viene chiamato Registro di Indirizzo. Il bit 7 di $D600 è il Bit di Stato Update Ready (di Pronto Aggiornamento). Ad una scrittura in esso, i cinque bit meno significativi convogliano l'indirizzo di un registro interno per accedervi in qualche modo. Ad una lettura di questo registro, viene riportato un byte di stato. Il bit 7 di questo registro è basso mentre viene aggiornata la memoria di visualizzazione e diventa alto quando è pronto per l'operazione successiva. Il sesto bit ritornerà basso per una condizione non valida del registro della penna ottica ed alto per un indirizzo valido della penna ottica. Il registro finale indica con basso che l'analisi non avviene in soppressione verticale e con un alto che è in soppressione verticale.

L'altro registro è il Registro Dati. Può essere letto e vi si può scrivere. Il suo scopo è di scrivere i dati nel registro interno selezionato dal registro di indirizzo. In tutti i registri interni si può leggere e scrivere attraverso questo registro, sebbene non tutti siano di 8 bit di ampiezza.

REGISTRI INTERNI

Ci sono 37 registri interni nell'8563, usati per unà varietà di operazioni. Essi si dividono in due gruppi di base: registri per la predisposizione e registri di visualizzazione. I registri di predisposizione sono usati per definire conti interni per una visualizzazione corretta. Variando questi registri, l'utente può configurare l'8563 per l'NTSC, il PAL o altri video standard. I registri di visualizzazione sono usati per definire e gestire i caratteri sullo schermo. Una volta che un insieme di caratteri è stato caricato in questo chip, è possibile visualizzare un testo di 80 colonne a colori digitali di 4 bit. Ci sono anche comandi per il movimento dei blocchi che rimuovono il tempo overhead (assorbito dal sistema operativo per la gestione delle operazioni di multiprogrammazione oltre all'esecuzione dei programmi stessi) di cui si ha bisogno per caricare una grossa quantità di dati nel chip attraverso i due livelli di indirizzamento indiretto. La Figura 17-11 è una visualizzazione della mappa del registro interno dell'8563.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R00	Totale Orizzontale							
R01	Orizzontale Visualizzato							
R02	Posizione di Sinc Orizzontale							
R03	Ampiezza di Sinc Verticale				Ampiezza di Sinc Orizzontale			
R04	Totale Verticale							
R05				Aggiustamento Totale Verticale				
R06	Verticale Visualizzato							
R07	Posizione di Sinc Verticale							
R08							Modo Intercalato	
R09				Carattere Totale Verticale				
R10		Modo Cursore		Linea Inizio Scansione del Cursore				
R11				Linea Fine Scansione del Cursore				
R12	Indirizzo Inizio Video (Alto)							
R13	Indirizzo Inizio Video (Basso)							
R14	Posizione Cursore (Alta)							
R15	Posizione Cursore (Bassa)							
R16	Penna Ottica Verticale							
R17	Penna Ottica Orizzontale							
R18	Locazione di Aggiornamento (Alta)							
R19	Locazione di Aggiornamento (Bassa)							
R20	Indirizzo Inizio Attributo (Alto)							
R21	Indirizzo Inizio Attributo (Basso)							
R22	Carattere Totale Orizzontale				Carattere Visualizzato Orizzontale			
R23				Carattere Visualizzato Verticale				
R24	Copia/Riempi	Rev Schermo	Intervallo Blink	Scroll Dolce Verticale				
R25	Grafica/Testo	Abil Attrib	Semigraf	Pix Dbl	Scroll Dolce Orizzontale			
R26	Colore di Primo Piano				Colore di Fondo			
R27	Incremento dell'Indirizzo per Riga							
R28	Indirizzo Pos Carattere			4164/4416				
R29				Linea di Scansione Sottolineatura				
R30	Conto Parole (conto-1)							
R31	Lettura/Scrittura Dati CPU							
R32	Indirizzo Sorgente della Copia Blocco (Alto)							
R33	Indirizzo Sorgente della Copia Blocco (Basso)							
R34	Inizio Abilitazione Video							
R35	Fine Abilitazione Video							
R36					Ricarica DRAM per la Linea di Scansione			

Figura 17-11. Mappa del Registro 8563

DESCRIZIONE DEI SEGNALI

Sono coinvolti molti segnali diversi nel chip 8563, ma si possono dividere in tre categorie generali. I segnali dell'interfaccia del CPU servono da interfaccia verso il bus 8502. I segnali di gestione del bus locale servono per mantenere il bus della memoria locale. Infine, i segnali di interfaccia del video sono i segnali necessari per fornire un'immagine RGBI su un monitor RGBI. La configurazione dei pin dell'8563 viene mostrata nella Figura 17-12.

L'INTERFACCIA DEL CPU

Il chip 8563 interfaccia con il bus 8502 usando il minimo dei segnali. Questo è dovuto in modo principale alla memoria locale usata dall'8563. I segnali dell'interfaccia del CPU sono i seguenti:

- **D_0-D_7**: Bus dei dati bidirezionale che permette ai dati di passare tra l'8563 e l'8502. Si trova ai pin da 18 a 13, 11 e 10.
- **CS**: Input di selezione del chip. Questo input deve essere alto per la selezione e le operazioni corrette del chip. Localizzato al pin 4.
- **/CS**: Input di selezione del chip. Questo input deve essere basso per la selezione e le operazioni corrette del chip. Localizzato al pin 7.
- **/RS**: Input per la selezione del Registro. Alto permette le letture e le scritture nel registro dati selezionato. Basso permette letture del registro di stato e scritture nel registro di indirizzo. Nel sistema, questa linea è legata a A0. È localizzato al pin 8.
- **R/W**: Questa linea controlla la direzione dei dati per il bus dei dati. È un tipico segnale di controllo dell'8502. Si trova al pin 9.
- **INIT**: Input attivo basso per azzerare i latch di controllo interni, permettendo al chip di iniziare il funzionamento seguendo l'accensione iniziale. È connesso a /RES nel C128, al pin 23.
- **DISPEN**: Segnale di output dell'Abilitazione di Visualizzazione, non usato nel C128. Si trova al pin 19.
- **RES**: Questo input inizializza tutti i contatori dell'analisi interna, ma non i registri di controllo. Non è usato attivamente nel circuito del C128 e non si trova al pin 22.
- **TST**: Pin usato solo per le prove, collegato alla terra nel C128. Localizzato al pin 24 del chip.

L'INTERFACCIA PER LA GESTIONE DEL BUS LOCALE

L'interfaccia per la gestione del bus locale è un gruppo di segnali generati dall'8563 per la gestione del DRAM del video locale. Questo DRAM locale semplifica l'aggiunta di un visualizzatore ad 80 colonne ad un sistema ed abilita un computer del sistema con uno spazio limitato dell'indirizzo per sopportare un video ad 80 colonne senza mettere a dura prova le sue risorse limitate di memoria.

- **DD_0-DD_7**: Bus dei dati del DRAM bidirezionale locale di visualizzazione, comprendente i pin 35-36 e 38-42.
- **DA_0-DA_7**: Bus di indirizzo del DRAM locale multiplex di visualizzazione. Occupa i pin 26-33.
- **DR/W**: Lettura/Scrittura del DRAM locale di visualizzazione, pin 21.
- **/RAS**: Impulso stroboscopico dell'indirizzo di riga per il DRAM locale, pin 47.
- **/CAS**: Indirizzo stroboscopico dell'indirizzo di colonna per il DRAM locale, pin 48.

L'INTERFACCIA DEL VIDEO

L'insieme finale dei segnali dell'8563 sono i segnali dell'interfaccia del video. Questi segnali sono connessi in modo diretto all'immagine visualizzata dal video.

- **DCLK**: Dot Clock del Video, determina la larghezza dei pixel e viene usato internamente come base di temporizzazione per tutti i segnali sincronizzati, come il carattere clock e la temporizzazione DRAM. Si trova al pin 2.
- **CCLK**: Output del carattere clock, non usato nel sistema C128 e si trova al pin 1.
- **LP2**: Input per la penna ottica; una condizione di transizione positiva su questo input fà da latch per la posizione verticale ed orizzontale del carattere visualizzato in quel momento. Si trova al pin 25.
- **HSYNC**: Segnale di sincronizzazione orizzontale, programmabile interamente attraverso i registri interni dell'8563 e si trova al pin 20.
- **R,G,B,I**: Output dei dati in pixel. Formano un codice di 4 bit associato ad ogni pixel, contenente le informazioni del colore e dell'intensità, fino ad un totale di 16 colori o ombre grigie visualizzate. Localizzato ai pin 46, 45, 44, 43, rispettivamente.

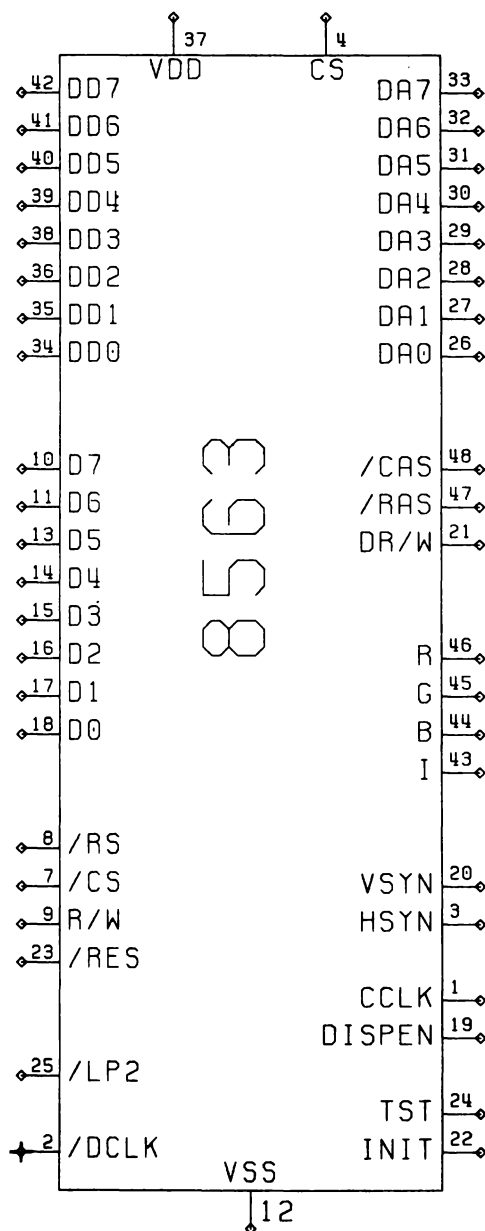



Figura 17-12. Configurazione dei pin del Chip 8563

SPECIFICHE DEL CHIP PER L'UNITÀ INTERFACCIA DEL SUONO (SID) 6581

CONCETTO

L'Unità Interfaccia del Suono 6581 (SID) è un chip singolo, sintetizzatore della musica elettronica a tre voci/generatore degli effetti sonori, compatibile con l'8502 e con le famiglie simili di microprocessori. Il SID fornisce un controllo del tono (frequenza) ad ampio raggio, alta risoluzione, il colore del tono (contenuto armonico) e le onde dinamiche (volume). Circuiti elettronici specializzati per il controllo, minimizzano l'overhead del software, facilitando l'uso nei video game arcade e domestici e strumenti musicali a basso costo.

FUNZIONI

- 3 OSCILLATORI DEL TONO
  Intervallo: 0-4 kHz
- 4 FORME D'ONDA PER OSCILLATORE
  Triangolo, Dente di Sega, Impulso
  Variabile, Rumore
- 3 MODULATORI DELL'AMPIEZZA
  Intervallo: 48 dB
- 3 GENERATORI ENVELOPE
  Risposta Esponenziale
  Intervallo Attack: 2ms-8 s
  Intervallo Decay : 6ms-24 s
  Valore di Sustain: 0-massimo del vol
  Intervallo Release: 6ms-24 s
- SINCRONIZZAZIONE DELL'OSCILLATORE
- MODULAZIONE DEL SUONO
- FILTRO PROGRAMMABILE
  Intervallo di Cutoff: 30 Hz-12 kHz
  12 dB/Rolloff dell'ottava
  Passa Basso, Passa Banda,
  Passa Alto, Output della Tacca
  Risonanza Variabile

- CONTROLLO DEL VOLUME PRINCIPALE
- 2 A/D INTERFACCE POT
- NUMERO CASUALE/GENERATORE DI MODULAZIONE
- INPUT DELL'AUDIO ESTERNO

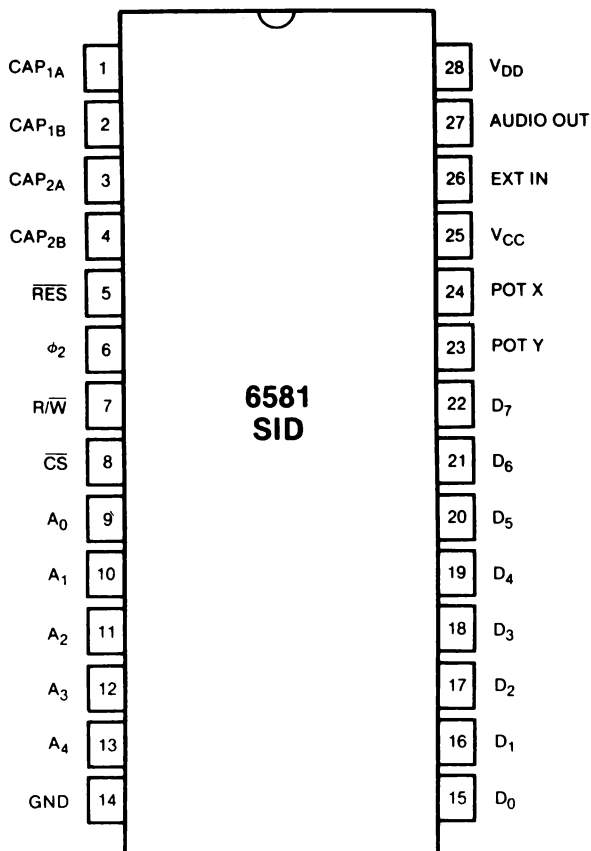



Figura 17-13. Configurazione dei pin SID 6581

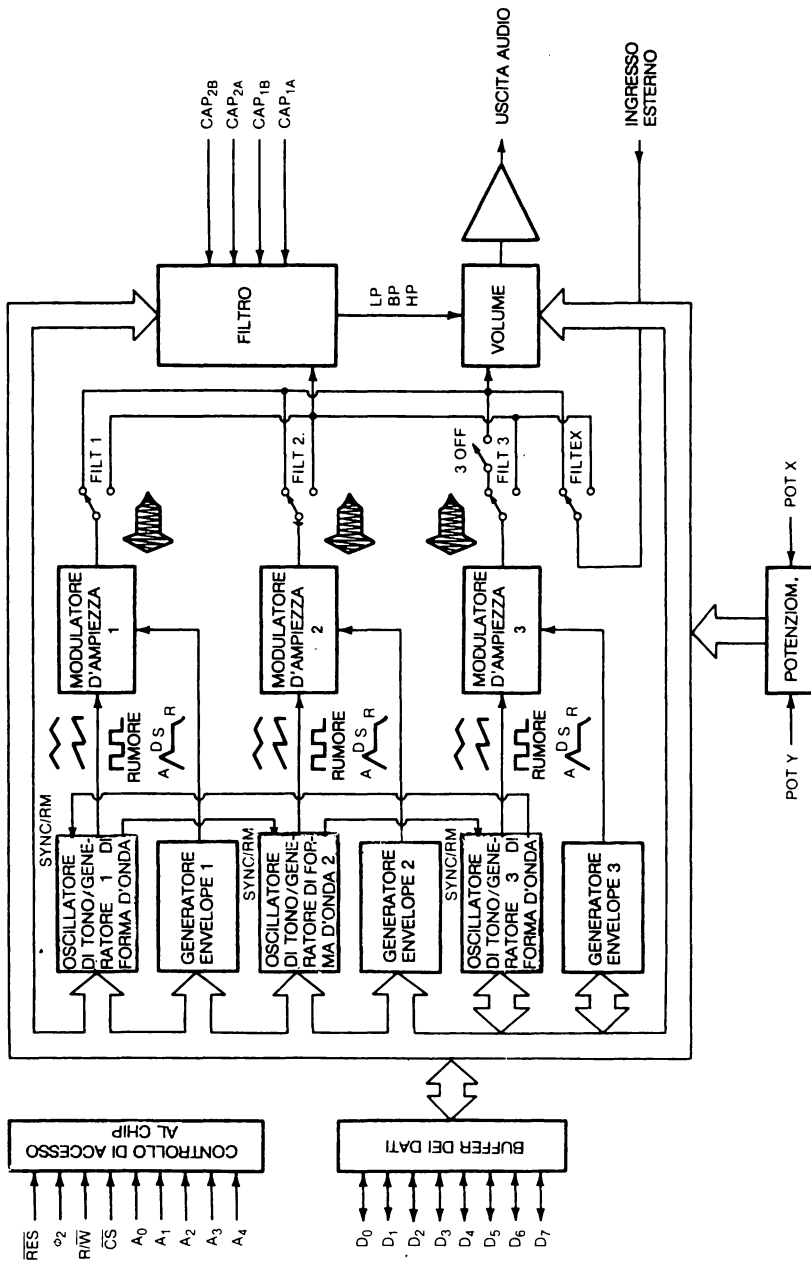



Figura 17-14. Diagramma a blocchi del 6581

DESCRIZIONE

Il 6581 consiste di tre sintetizzatori "voci" che possono essere usati in modo indipendente o in connessione l'un con l'altro (o con fonti audio esterne) per creare suoni complessi. Ogni voce consiste di un Oscillatore del Tono/Generatore della Forma d'Onda, di un Generatore Envelope e di un Modulatore dell'Ampiezza. L'Oscillatore del Tono controlla il tono della voce superiore ad una certa ampiezza di intervallo. L'Oscillatore produce quattro forme d'onda alla frequenza selezionata, con l'unico contenuto armonico di ogni forma d'onda che fornisce un controllo semplice del colore del tono. Le dinamiche del volume dell'oscillatore sono controllate dal Modulatore dell'Ampiezza sotto la direzione del Generatore Envelope. Se è triggerato, il Generatore Envelope crea un'ampiezza envelope con intervalli programmabili di aumento e diminuzione del volume. Oltre alle tre voci, viene fornito un Filtro programmabile per generare colori del tono complessi e dinamici per mezzo di una sintesi sottrattiva.

Il SID permette al microprocessore di leggere l'output che cambia del terzo Oscillatore e del terzo Generatore Envelope. Questi output possono essere usati come fonte per la modulazione dell'informazione per creare il vibrato, sweep tra frequenza e filtro ed effetti simili. Il terzo oscillatore può agire anche come generatore di numeri casuali per i giochi. Due convertitori A/D sono forniti per interfacciare il SID con i potenziometri. Questi possono essere usati per i "paddle" in un ambiente di gioco o come pannello di fonte per il controllo di un sintetizzatore musicale. Il SID può esegiure segnali audio esterni, permettendo ai chip SID multipli di essere concatenati o mischiati in sistemi polifonici complessi. Per una descrizione completa dei registri, consultate il Capitolo 12, Suono e Musica sul Commodore 128.

DESCRIZIONE DEI PIN SID

CAP1A, CAP1B, (PIN 1,2)/CAP2A, CAP2B (PIN 3,4)

Questi pin sono usati per connettere i due condensatori integratori richiesti dal filtro programmabile. C1 connette i pin 1 e 2, C2 i pin 3 e 4. Entrambi i condensatori dovrebbero avere lo stesso valore. Un normale funzionamento del filtro sull'intervallo audio (approssimativamente 30 Hz-122 kHz) è ottenuto con un valore di 2200 pF per C1 e C2. I condensatori di polistirene sono preferibili e in sistemi polifonici complessi, dove molti chip SID devono seguire l'un l'altro, sono consigliabili i condensatori corrispondenti.

L'intervallo di frequenza del filtro può essere adattato ad applicazioni specifiche con la scelta di valori del condensatori. Per esempio, un gioco a basso costo può non richiedere una risposta a piena alta frequenza. In questo caso, si potrebbero scegliere valori larghi per C1 e C2 per dare un maggiore controllo delle frequenze profonde del filtro. La frequenza massima di cutoff del filtro è data da:

$FC_{max} = 2.6E-5/C$

dove C è il valore del condensatore. L'intervallo del filtro si estende 9 ottave sotto la frequenza massima di cutoff.

RES (PIN 5)

Questo input del valore TTL è il reset di controllo per il SID. Se diventa basso per almeno 10 cicli Φ, tutti i registri interni sono resettati a 0 e l'output audio è disonorizzato. Questo pin di solito è connesso alla linea di reset del microprocessore o ad un circuito azzeramento dell'accensione (power-on-clear).

Φ2 (PIN 6)

Questo input del valore TTL è il clock principale del SID. Tutte le frequenze dell'oscillatore e gli intervalli envelope fanno riferimento a questo clock. Φ2 controlla anche i trasferimenti dei dati tra il SID ed il microprocessore. I dati possono essere trasferiti solamente quando Φ2 è alto. Essenzialmente, Φ2 agisce da selezionatore del chip alto attivo per quanto concerne il trasferimento dei dati. Questo pin di solito è connesso al clock del sistema, con una frequenza di funzionamento nominale di 1.0 MHz.

R/W (PIN 7)

Questo input del valore TTL controlla la direzione dei trasferimenti dei dati tra il SID ed il microprocessore. Se sono state incontrate le condizioni di selezione del chip, un valore alto di questa linea permette al microprocessore di Leggere i dati dal registro SID selezionato ed uno basso permette al microprocessore di Scrivere i dati nel registro SID selezionato. Questo pin di solito è connesso alla linea Lettura/Scrittura del sistema.

CS (PIN 8)

Questo input del valore TTL è una selezione bassa attiva del chip che controlla i trasferimenti dei dati tra il SID ed il microprocessore. CS deve essere basso per qualsiasi trasferimento. Una Lettura dal registro SID selezionato può aver luogo solo se CS è basso, Φ2 è alto e R/W è alto. Una Scrittura nel registro SID selezionato può avvenire solo se CS è basso, Φ2 è alto e R/W è basso. Questo pin di solito è connesso ai circuiti di decodificazione dell'indirizzo, permettendo al SID di risiedere nella mappa di memoria di un sistema.

A0-A4 (PIN 9-13)

Questi input dei valori TTL sono usati per selezionare uno dei 29 registri SID. Sebbene siano forniti sufficienti indirizzi per selezionare uno dei 32 registri, le rimanenti tre locazioni dei registri non sono usate. Una Scrittura in una qualsiasi di queste tre locazioni viene ignorata ed una Lettura riporta un dato non valido.

Questi pin sono di solito connessi con le linee di indirizzo corrispondenti del microprocessore così che ci si può indirizzare al SID nello stesso modo della memoria.

GND (PIN 14)

Per risultati migliori, la linea di terra tra il SID e l'alimentazione della potenza dovrebbe essere separata dalle linee di terra in altri circuiti digitali. Questo renderà minimo il disturbo digitale nell'output audio.

D0-D7 (PIN 15-22)

Queste linee bidirezionali sono usate per trasferire i dati tra il SID e il microprocessore. Esse sono TTL compatibili nel modo input e sono in grado di pilotare due caricamenti TTL nel modo output. I buffer dei dati sono di solito in stato di alta impedenza spenta. Durante un'operazione di Scrittura, i buffer dei dati rimangono nello stato spento (input) ed il microprocessore fornisce i dati al SID su queste linee. Durante un'operazione di Lettura, i buffer dei dati si accendono ed il SID fornisce i dati al microprocessore su queste linee. I pin di solito sono connessi alle linee dei dati corrispondenti del microprocessore.

POTX, POTY (PIN 24,23)

Questi pin sono input verso i convertitori A/D usati per digitalizzare la posizione dei potenziometri. Il processo di conversione è basato sulla costante del tempo di un condensatore collegato a terra dal pin POT, caricato da un potenziometro legato dal pin POT a +5 volt. I valori del componente sono determinati da:

RC = 4.7 E-4

dove R è la resistenza massima del pot e C è il condensatore.
Maggiore è il condensatore, minore è il valore jitter POT. I valori consigliati di R e C sono 470 kΩ e 1000 pF. Notate che sono richiesti potenziometro e condensatore separati per ciascun pin POT.

V_{cc} (PIN 25)

Come per la linea GND, dovrebbe essere fatta girare una linea separata +5 VDC tra il SID Vcc e l'alimentazione per minimizzare il disturbo. Un condensatore bypass dovrebbe essere localizzato vicino al pin.

EXT IN (PIN 26)

Questo input analogo permette ai segnali esterni audio di essere mischiati con l'output audio del SID o di essere passati attraverso il Filtro. Sorgenti tipiche includono la voce, la chitarra e l'organo. L'impedenza di input di questo pin è

dell'ordine di 100 kΩ. Qualsiasi segnale applicato direttamente al pin dovrebbe andare ad un livello DC di 6 volt e non eccedere 3 volt p-p. Per prevenire qualsiasi interferenza causata dalle differenze di livello DC, dei segnali esterni dovrebbero essere accoppiati AC ad EXT IN da un condensatore elettrolitico nell'intervallo 1-10 μf. Poichè il percorso audio diretto (FILTEX=0) ha un guadagno di unità, EXT IN può essere usato per mischiare gli output di chip SID con la concatenazione. Il numero di chip che possono essere concatenati in questo modo è determinato dalla quantità di disturbo e distorsione permessa come output finale. Notate che il controllo del volume di output coinvolgerà non solo le tre voci SID, ma anche qualsiasi input esterno.

AUDIO OUT (PIN 27)

Questo buffer a sorgente aperta è l'output audio finale del SID, comprese le tre voci SID, il filtro e qualsiasi input esterno. Il valore di output è posto dal controllo dell'output del volume e raggiunge un massimo di 2 volt p-p ad un valore DC di 6 volt. Un resistore della sorgente dall'Audio Out a terra è richiesto per un funzionamento adeguato. La resistenza consigliata è 1kΩ per un'impedenza output standard.

Poichè l'output del SID va ad un valore DC di 6 volt, AC dovrebbe essere accoppiato a qualsiasi amplificatore audio con un condensatore elettrolitico nell'intervallo 1-10 μf.

V_{dd} (PIN 28)

Come per V_{cc}, dovrebbe essere fatta girare una linea +12 VDC separata verso SID V_{DD} e dovrebbe venire usato un condensatore bypass.

CARATTERISTICHE SID 6581

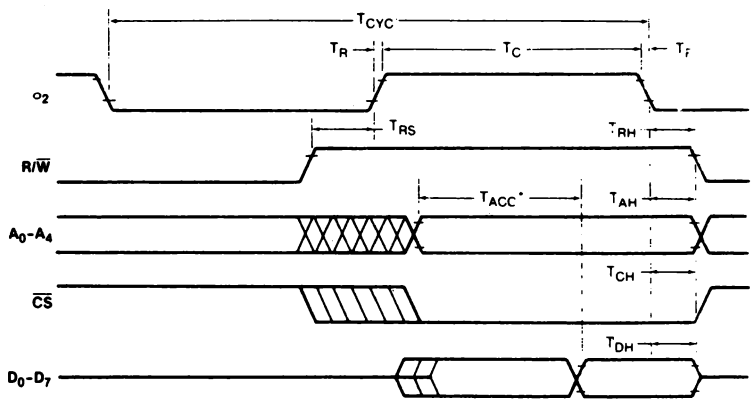
VALORI MASSIMI ASSOLUTI

VALORE	SIMBOLO	VALORE	UNITÀ
Voltaggio di Alimentazione	V_{DD}	da -0.3 a +17	VDC
Voltaggio di Alimentazione	V_{cc}	da -0.3 a +7	VDC
Voltaggio di Input (analogico)	V_{ina}	da -0.3 a +17	VDC
Voltaggio di Input (digitale)	V_{ind}	da -0.3 a +7	VDC
Temperatura di Funzionamento	T_A	da 0 a 70	°C
Temperatura di Memorizzazione	T_{STG}	da -55 a +150	°C

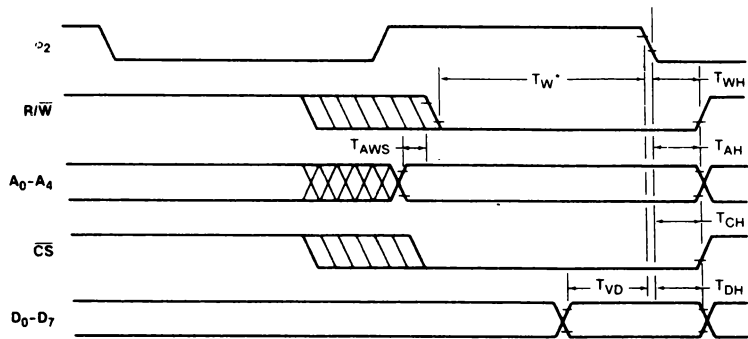
CARATTERISTICHE ELETTRICHE

(V_{DD} = 12 VDC ± 5%, V_{cc} = 5 VDC ± 5%, T_A = 0 da 0 a 70 C)

CARATTERISTICHE		SIMBOLO	MIN	TIPO	MAX	UNITÀ
Voltaggio Alto In Input	(RES, Φ2, R/W, CS	V_{IH}	2	-	V_{cc}	VDC
Voltaggio Basso In Input	A0-A4, D0-D7)	V_{IL}	-0.3	-	0.8	VDC
Corrente di fuga In Input	(RES, Φ2, R/W, CS, A0-A4; V_{in} = 0-5VDC)	I_{in}	-	-	2.5	μA
Tri-Stati (Off) Corrente di fuga In Input	(D0-D7; V_{cc} = max) V_{in} = 0.4-2.4 VDC	I_{TSI}	-	-	10	μA
Voltaggio Alto In Output	(D0-D7; V_{cc} = min, 1 load = 200 μA	V_{OH}	2.4	-	V_{cc}-0.7	VDC
Voltaggio Basso In Output	(D0-D7; V_{cc} = max, 1 load = 3.2 mA	V_{OL}	GND	-	0.4	VDC
Corrente Alta Output	(D0-D7; Sourcing, V_{OH} = 2.4 VDC	I_{OH}	200	-	-	μA
Corrente Bassa Output	(D0-D7; Sinking V_{OL} = 0.4 VDC	I_O	3.2	-	-	VDC
Capacità In Input	(RES, Φ2, R/W, CS A0-A4, D0-D7)	C_{in}	-	-	10	pF
Voltaggio del Potenziatore Trigger	(POTX, POTY)	V_{pot}	-	V_{cc}/2	-	VDC
Potenziatore della Corrente di Sincronismo	(POTX, POTY)	I_{pot}	500	-	-	μA
Impedenza In Input	(EXT IN)	R_n	100	150	-	kΩ
Voltaggio di Input Audio	(EXT IN)	V_{in}	5.7	6	6.3	VDC
Voltaggio di Output Audio	(AUDIO OUT; 1 kV load, volume = max)	V_{out}	5.7	6	6.3	VDC
	Una voce su:		0.4	0.5	0.6	VAC
	Tutte le voci su:		1.0	1.5	2.0	VAC
Corrente di Alimentazione	(V_{DD})	I_{DD}	-	20	25	mA
Corrente di Alimentazione	(V_{cc})	1_{cc}	-	70	100	mA
Dissipazione di Potenza	(Totale)	P_D	-	600	1000	mW

TEMPORIZZAZIONE SID 6581

*TACC viene misurato dall'ultima occorrenza di Φ2, CS, A0-A4.

SIMBOLO	NOME	MIN	TIPO	MAX	UNITÀ
T_{CYC}	Tempo Ciclo Clock	1	-	20	µs
T_C	Ampiezz. Impulso Alto Clock	450	500	10,000	ns
T_R, T_F	Tempo di salita/caduta del clock	-	-	25	ns
T_{RS}	Legge Tempo di Setup	0	-	-	ns
T_{RH}	Legge Tempo di Tenuta	0	-	-	ns
T_{ACC}	Tempo di Accesso	-	-	300	ns
T_{AH}	Tempo di Tenuta Indirizzo	10	-	-	ns
T_{CH}	Tempo di Tenuta Selez. Chip	0	-	-	ns
T_{DH}	Tempo di Tenuta Dati	20	-	-	ns

Figura 17-15. Ciclo di lettura

*TW viene misurato dall'ultima occorrenza di Φ2, CS, R/W.

SIMBOLO	NOME	MIN	TIPO	MAX	UNITÀ
T_W	Ampiezz. Impulso Scritt.	300	-	-	ns
T_{WH}	Tempo Tenuta Scritt.	0	-	-	ns
T_{AWS}	Tempo Set-up Indirizz.	0	-	-	ns
T_{AH}	Tempo Tenuta Indirizz.	10	-	-	ns
T_{CH}	Tempo Tenuta Selez. Chip	0	-	-	ns
T_{VD}	Dato Valido	80	-	-	ns
T_{DH}	Tempo Tenuta Dati	10	-	-	ns

Figura 17-16. Ciclo di Scrittura

VALORI DELLA SCALA MUSICALE A TEMPERAMENTO EQUABILE

La Tabella del Capitolo 12 lista i valori numerici che devono essere memorizzati nei registri SID di controllo della frequenza dell'Oscillatore per produrre le note della scala musicale a temperamento equabile. La scala a temperamento equabile consiste di un'ottava contenente 12 semitoni (note): C,D,E,F,G,A,B e C#,D#,F#,G#,A#. La frequenza di ogni semitono è esattamente la radice dodicesima di 2 (2) volte la frequenza del semitono precedente. La tabella è basata su un clock Φ2 di 1.02 MHz. Riferitevi all'equazione data nella Descrizione del Registro nel Capitolo 12 per l'uso di altre frequenze del clock principale. La scala selezionata è altezza da concerto, nella quale A–4 = 440 Hz. Sono possibili anche trasposizioni di questa scala e di scale diverse dalla scala a temperamento equabile.

Sebbene la tabella del Capitolo 12 fornisca un metodo semplice e veloce per generare la scala a temperamento equabile, ciò è inefficiente dal punto di vista della memoria perchè richiede 192 byte solo per la tabella. L'efficienza della memoria può essere migliorata determinando il valore della nota con un algoritmo. Utilizzando il fatto che ogni nota di un'ottava ha esattamente metà della frequenza di quella nota nell'ottava successiva, la tabella migliorata delle note può essere ridotta da 96 entrate a 12 entrate, poichè ci sono 12 note ogni ottava. Se le 12 entrate (24 byte) consistono dei valori a 16 bit dell'8ava ottava (C-7 fino a B-7), allora le note delle ottave più basse si possono derivare scegliendo la nota appropriata nell'8ava ottava e dividendo il valore a 16 bit per due per ogni ottava di differenza. Poichè la divisione per due non è altro che uno spostamento a destra del valore, si può eseguire facilmente il calcolo con una semplice routine software. Sebbene la nota B-7 sia oltre l'intervallo degli oscillatori, questo valore potrebbe essere ancora incluso nella tabella a scopo di calcolo (MSB di B-7 richiederebbe un contenitore software speciale, come la generazione di questo bit nel CARRY prima dello spostamento). Ogni nota deve essere specificata in forma che indichi quale dei 12 semitoni è desiderato ed in quale delle 8 ottave è il semitono. Poichè sono necessari 4 bit per selezionare uno dei 12 semitoni e tre bit sono necessari per selezionare una delle 8 ottave, l'informazione si può adattare ad un byte, con il nybble più basso che seleziona il semitono (indirizzandosi alla tabella migliorata) ed il nybble più alto che viene usato dalla routine della divisione per determinare quante volte il valore della tabella deve essere spostato a destra.

GENERATORI ENVELOPE SID

Il generatore envelope in quattro parti ADSR (ATTACK, DECAY, SUSTAIN, RELEASE) è stato collaudato in musica elettronica per fornire uno scambio (trade-off) ottimo tra flessibilità e facilità del controllo dell'ampiezza. Una selezione appropriata dei parametri envelope permette la simulazione di un grosso numero di strumenti a percussione e sostenuti. Il violino è un buon esempio di strumento sostenuto. Il violinista controlla il volume piegando lo strumento. In modo tipico, il volume si forma lentamente, raggiunge il massimo e poi cade ad un livello intermedio. Il violinista può mantenere questo livello per quanto desidera, poi il volume muore lentamente. Un''istantanea'' di questo envelope viene mostrata nella figura 17-17.

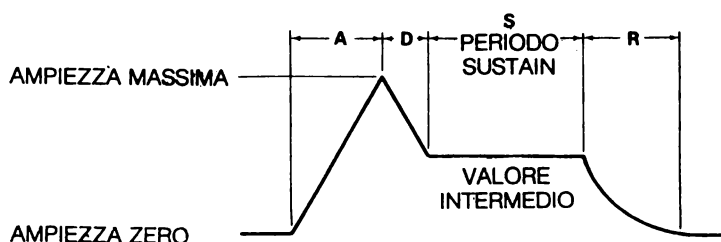
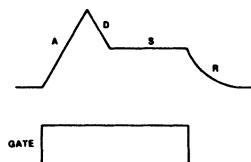
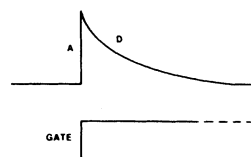



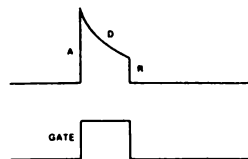
Figura 17-17. Envelope ADSR

Questo envelope del volume può essere riprodotto facilmente dall'ADSR come mostrato sotto, con valori tipici envelope:

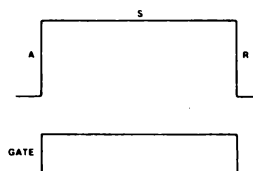
ATTACK:	**10 ($A)**	**500 ms**
DECAY:	**8**	**300**
SUSTAIN:	**10 ($A)**	
RELEASE:	**9**	**750 ms**

Notate che il tono può essere tenuto nel livello immediato SUSTAIN per quanto si desidera. Il tono non morirà finchè GATE non viene cancellato. Con minori alterazioni, questo envelope di base può essere usato per l'ottone e gli strumenti a fiato come per quelli a corda.

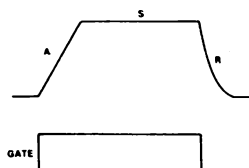
Una forma envelope completamente diversa viene prodotta dagli stumenti a percussione tipo i tamburi, i cembali ed i gong come da certe tastiere come pianoforti e clavicembali. L'envelope della percussione è caratterizzato da un attacco quasi istantaneo, seguito immediatamente da una caduta al volume zero. Gli stumenti a percussione non possono essere sostenuti ad un'ampiezza costante. Per esempio, al momento in cui un taburo viene colpito esso raggiunge il massimo del volume e cade rapidamente senza badare a come è stato colpito. Un envelope tipico del cembalo viene mostrato qui sotto:

ATTACK:	**0**	**2 ms**
DECAY:	**9**	**750 ms**
SUSTAIN:	**0**	
RELEASE:	**9**	**750 ms**

Notate che il tono comincia subito a decadere all'ampiezza zero dopo che si è raggiunto il massimo, senza tenere conto di quando GATE è azzerato. L'ampiezza dell'envelope dei pianoforti e dei clavicembali è in qualche modo più complicata, ma può essere generata facilmente con ADSR. Questi strumenti raggiungono il volume massimo quando un tasto viene colpito per primo. L'ampiezza incomincia a diminuire subito lentamente per il tempo in cui il tasto rimane schiacciato. Se il tasto viene rilasciato prima che il suono sia scomparso completamente, l'ampiezza cadrà immediatamente a zero. Questo envelope viene mostrato sotto:

ATTACK:	**0**	**2 ms**
DECAY:	**9**	**750 ms**
SUSTAIN:	**0**	
RELEASE:	**0**	**6 ms**

Notate che il tono cade lentamente finchè GATE non viene azzerato, momento in cui l'ampiezza cade rapidamente a zero.

L'envelope più semplice è quello dell'organo. Quando viene premuto un tasto, il tono raggiunge subito il volume massimo e rimane là. Quando il tasto è rilasciato, il tono cade subito al volume zero. Questo envelope viene mostrato sotto:

ATTACK:	**0**	**2 ms**
DECAY:	**0**	**6 ms**
SUSTAIN:	**15($F)**	
RELEASE:	**0**	**6 ms**

Il potere reale del SID stà nell'abilità di creare suoni originali piuttosto che simulazioni degli strumenti musicali. ADSR è in grado di creare envelope che non corrispondono a nessuno strumento "reale". Un buon esempio è l'envelope "all'indietro". Questo envelope è caratterizzato da un attacco lento e da una caduta rapida che suona come uno strumento che è stato registrato su nastro e poi suonato all'incontrario. Questo envelope viene mostrato sotto:

ATTACK:	**10($A)**	**500 ms**
DECAY:	**0**	**6 ms**
SUSTAIN:	**15($F)**	
RELEASE:	**3**	**72 ms**

Molti suoni unici possono essere creati applicando l'ampiezza envelope di uno strumento alla struttura armonica di un alltro. Questo produce suoni simili a strumenti acustici familiari, ma molto diversi. In generale, il suono è molto soggettivo e saranno necessari esperimenti con vari valori envelope e contenuti armonici per ottenere il suono desiderato.

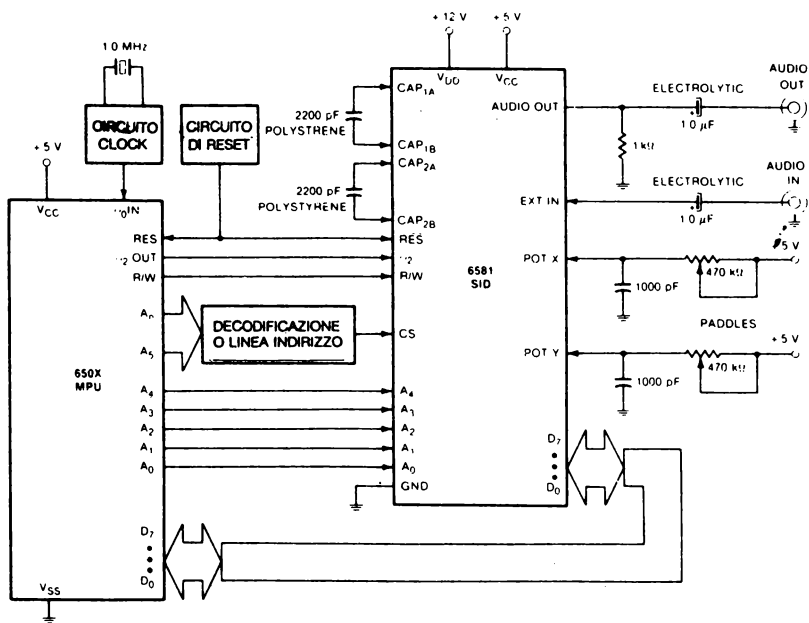



Figura 17-18 Applicazione tipica SID/6581

SPECIFICHE DEL CHIP (CIA) ADATTATORE AD INTERFACCIA COMPLESSA 6526

DESCRIZIONE

L'Adattatore ad Interfaccia Complessa (CIA) del 6526 è un'unità periferica compatibile con il bus 8502 con una temporizzazione e capacità di I/O estremamente flessibili. La Figura 17-19 mostra la configurazione dei pin del 6526. La Figura 17-20 mostra il diagramma a blocchi del 6526.

FUNZIONI

- 16 linee di I/O programmabili individualmente
- Handshaking a 8 o 16 bit di lettura o scrittura
- Due temporizzatori indipendenti a 16 bit collegabili
- Clock 24 ore (AM/PM) per l'ora del giorno con segnale acustico programmabile
- Registro shift ad 8 bit per I/O seriale
- Capacità di caricamento 2TTL
- Linee di I/O CMOS compatibili
- Funzionamento disponibile a 1 o 2 MHz

INFORMAZIONI DI ORDINAMENTO

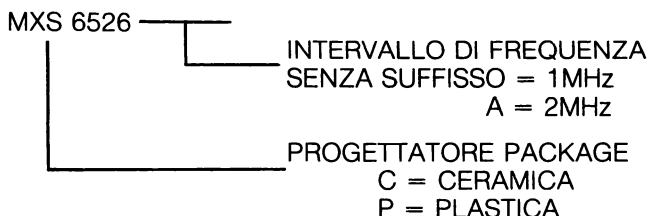
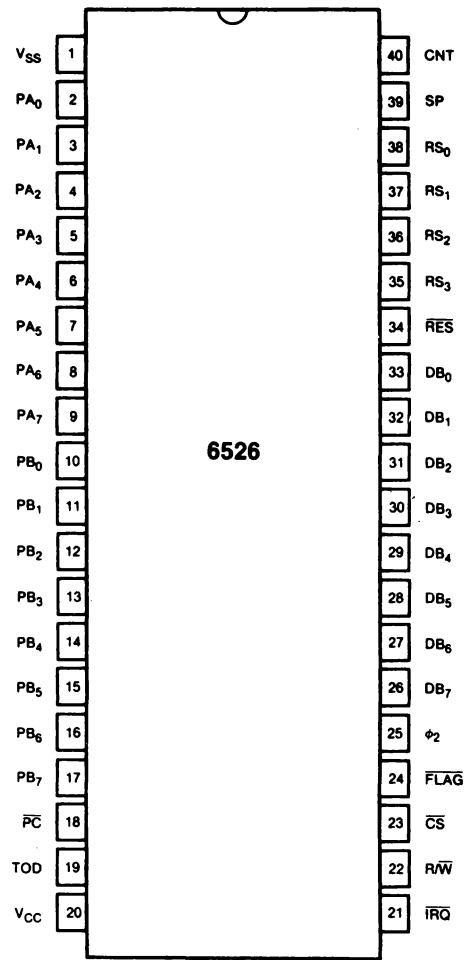



Figura 17-19. Configurazione dei pin CIA 6526

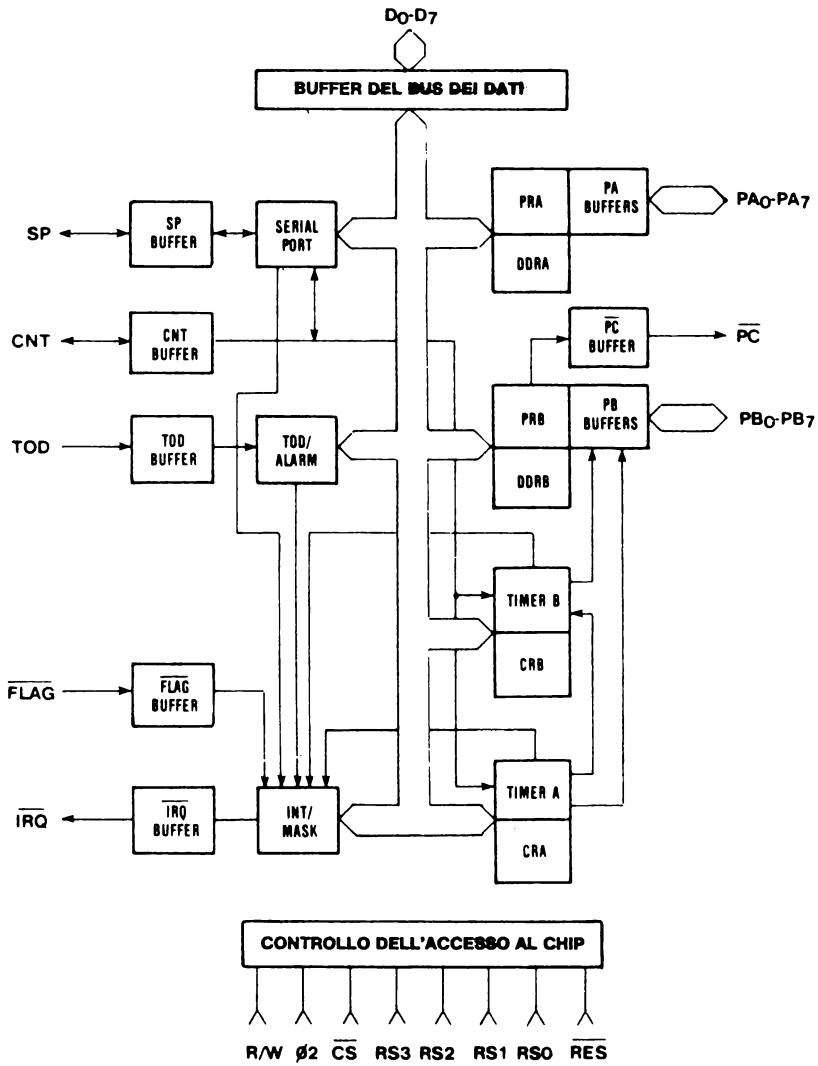



Figura 17-20. Diagramma a blocchi del 6526

VALORI MASSIMI ASSOLUTI

Voltaggio di Alimentazione V_{cc}	da -0.3 a +7.0V
Voltaggio di I/O, V_{IN}	da -0.3 a +7.0V
Temperatura di Funzionamento, T_{OP}	da 0 °C a 70 °C
Temperatura di Memorizzazione, T_{STG}	da 55 °C a 150 °C

Tutti gli input contengono dei circuiti di protezione per prevenire il danno dovuto a scariche elettrostatiche. Si dovrebbe fare attenzione a prevenire applicazioni non necessarie di voltaggio in eccesso rispetto ai limiti permessi.

COMMENTO

Tensioni oltre quelle elencate come "Valori massimi assoluti" possono causare un danno permanente dell'unità. Questi sono solo valori delle tensioni. Operazioni funzionali di quest'unità a queste o qualsiasi altra condizione oltre quelle indicate nelle sezioni operative di questa specifica non è implicata ed esposizioni a condizioni di valori massimi assoluti per periodi prolungati possono danneggiare l'affidabilità dell'unità.

CARATTERISTICHE ELETTRICHE

(V_{CC} + 5%, V_{SS} = 0 V, T_A = 0-70 °C)

CARATTERISTICHE	SIMBOLO	MIN	TIPO	MAX	UNITÀ
Voltaggio Alto in Input	V_{IH}	+2.4	-	V_{CC}	V
Voltaggio Basso in Input	V_{IL}	-0.3	-	-	V
Corrente di dispersione in Input $V_{IN} = V_{SS} +5V$ (TOD, R/W, $\overline{FLAG}$,Φ2, $\overline{RES}$, RS0-RS3, CS)	I_{IN}	-	1.0	2.5	μA
Resistenza di Prelievo dell'Input della porta	R_{PI}	3.1	5.0	-	KΩ
Corrente di dispersione in Output per lo stato di alta impendenza (tri-stati); V_{IN} = 4V a 2.4V; (DB0-DB7,SP,CNT,$\overline{IRQ}$)	I_{TSI}	-	±1.0	±10.0	μA
Voltaggio Alto in Output V_{CC} = MIN, $I_{LOAD} <$ -200μA (PA0-PA7,$\overline{PC}$ PB0-PB7,DB0-DB7)	V_{OH}	+2.4	-	V_{CC}	V
Voltaggio Basso in Output V_{CC} = MIN, $I_{LOAD} <$ 3.2mA	V_{OL}	-	-	+0.40	V
Corrente Alta di Output (Sorgente); $V_{OH} >$ 2.4V (PA0-PA7, PB0-PB7,PC,DB0-DB7	I_{OH}	-200	-1000	-	μA
Corrente Bassa di Output (Caduta); $V_{OL} >$ 4V (PA0-PA7,$\overline{PC}$ PB0-PB7,DB0-DB7	I_{OL}	3.2	-	-	mA
Capacità di Input	C_{IN}	-	7	10	pf
Capacità di Output	C_{OUT}	-	7	10	pf
Corrente di alimentazione	I_{CC}	-	70	100	mA

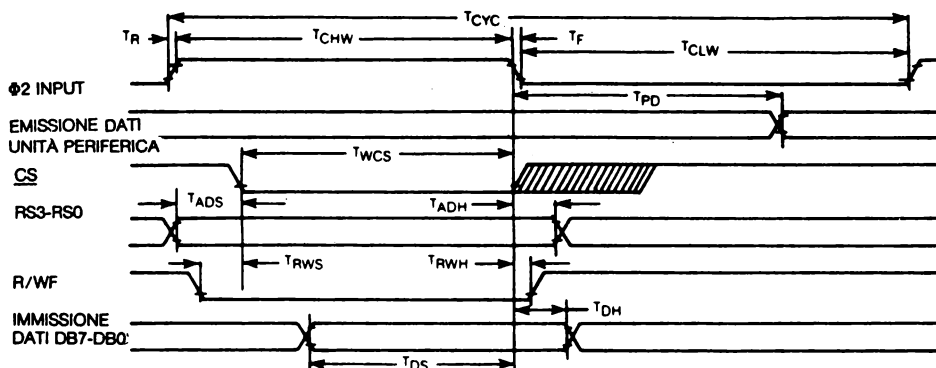



Figura 17-21 Diagramma di Scrittura della Temporizzazione

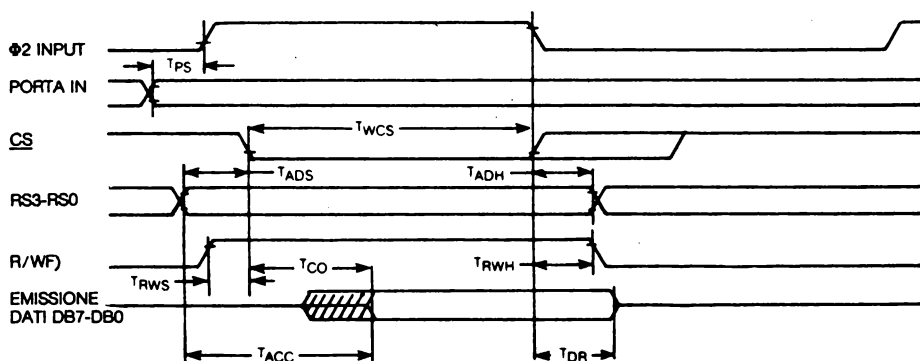



Figura 17-22 Diagramma di Lettura della Temporizzazione

SEGNALI DELL'INTERFACCIA 6526

Φ2—CLOCK DI INPUT

Il clock Φ2 è un input TTL compatibile usato per operazioni delle unità interne e come riferimento di temporizzazione per la comunicazione con il bus dati del sistema.

$\overline{CS}$—INPUT DI SELEZIONE DEL CHIP

L'input $\overline{CS}$ controlla l'attività del 6526. Un valore basso di $\overline{CS}$ mentre $\Phi 2$ è alto fà si che l'unità risponda ai segnali sull'R/W e sulle linee di indirizzo. Un alto di $\overline{CS}$ previene che queste linee controllino il 6526. La linea $\overline{CS}$ normalmente è attivata (bassa) a $\Phi 2$ dalla combinazione appropriata dell'indirizzo.

R/W—INPUT DI LETTURA/SCRITTURA

Il segnale R/W è di solito dato dal microprocessore e controlla la direzione del trasferimento dei dati del 6526. Un alto di R/W indica una lettura (trasferimento dei dati fuori del 6526), mentre un basso indica una scrittura (trasferimento dei dati nel 6526).

RS3-RS0—INPUT DELL'INDIRIZZO

Gli input dell'indirizzo selezionano gli indirizzi interni come descritto dalla Mappa dei Registri.

DB7-DB0—INPUT/OUTPUT DEL BUS DEI DATI

Gli otto pin del bus dei dati trasferiscono le informazioni tra il 6526 ed il bus dei dati del sistema. Questi pin sono input ad alta impedenza a meno che $\overline{CS}$ sia basso e R/W e $\Phi 2$ siano alti per la lettura dell'unità. Durante questa lettura, vengono abilitati i buffer degli output del bus dei dati, pilotando i dati dal registro selezionato al bus dei dati del sistema.

$\overline{IRQ}$—OUTPUT DELLA RICHIESTA DI INTERRUZIONE

$\overline{IRQ}$ è un output di scarico aperto normalmente connesso all'input di interruzione del processore. Un resistore esterno di prelievo delle informazioni tiene alto il segnale, permettendo agli output dell'$\overline{IRQ}$ multiplo di essere connessi. L'output $\overline{IRQ}$ normalmente è spento (alta impedenza) e viene attivato come basso come indicato nella descrizione del funzionamento.

$\overline{RES}$—INPUT RESET

Un basso del pin $\overline{RES}$ resetta tutti i registri interni. I pin della porta vengono posti come input ed i registri della porta a zero (sebbene una lettura delle porte li riporti tutti alti a causa dei prelevamenti passivi). I registri di controllo della temporizzazione sono posti a zero ed i latch della temporizzazione tutti a uno. Tutti gli altri registri sono resettati a 0.

CARATTERISTICHE DELLA TEMPORIZZAZIONE 6526

SIMBOLO	CARATTERISTICA	1 MHz MIN	1 MHz MAX	2 MHz MIN	2 MHz MAX	UNITÀ
	Φ2Clock					
T_{CYC}	**Tempo di ciclo**	1000	20,000	500	20,000	ns
$T_R T_F$	**Tempo di salita e Caduta**	-	25	-	25	ns
T_{CHW}	**Ampiezza dell'impulso del Clock (Alta)**	420	10,000	200	10,000	ns
T_{CLW}	**Ampiezza dell'impulso del Clock (Bassa)**	420	10,000	200	10,000	ns
	Ciclo di Scrittura					
T_{PD}	**Output del Ritardo da Φ2**	-	1000	-	500	ns
T_{WCS}	**$\overline{CS}$ basso mentre Φ2 alto**	420	-	200	-	ns
T_{ADS}	**Tempo di Predisposizione dell'indirizzo**	0	-	0	-	ns
T_{ADH}	**Tempo di Tenuta dell'Indirizzo**	10	-	5	-	ns
T_{RWS}	**Tempo di Predisposizione R/W**	0	-	0	-	ns
T_{RWH}	**Tempo di Tenuta R/W**	0	-	0	-	ns
T_{DS}	**Tempo di Predisposizione del Bus dei dati**	150	-	75	-	ns
T_{DH}	**Tempo di Tenuta del Bus dei Dati**	0	-	0	-	ns
	Ciclo di Lettura					
T_{PS}	**Tempo di Predisposizione della Porta**	300	-	150	-	ns
T_{WCS}(2)	**CS basso mentre 2 alto**	420	-	20	-	ns
T_{ADS}	**Tempo di Predisposizione dell'indirizzo**	0	-	0	-	ns
T_{ADH}	**Tempo di Tenuta dell'Indirizzo**	10	-	5	-	ns
T_{RWS}	**Tempo di Predisposizione R/W**	0	-	0	-	ns
T_{RWH}	**Tempo di Tenuta R/W**	0	-	0	-	ns
T_{ACC}	**Accesso ai Dati da RS3-RS0**	-	550	-	275	ns
I_{CO}(3)	**Accesso ai Dati da $\overline{CS}$**	-	320	-	150	ns
T_{DR}	**Tempo di Rilascio del Dato**	50	-	25	-	ns

1. Tutti i calcoli dei tempi fanno riferimento da V_{IL} massimo e V_{IH} minimo per gli input e V_{OL} massimo e V_{OH} minimo per gli output.

2. TWCS viene misurato dall'ultimo Φ2 alto o $\overline{CS}$ basso. $\overline{CS}$ deve essere basso almeno fino alla fine di Φ2 alto.

3. TCO è misurato dall'ultimo Φ2 alto o $\overline{CS}$ basso. I dati validi sono disponibili solo dopo l'ultimo T_{ACC} o T_{CO}.

MAPPA DEI REGISTRI

RS3	RS2	RS1	RS0	REG	NOME	DESCRIZIONE
0	0	0	0	0	PRA	REG A DATI UNITÀ PERIFERICHE
0	0	0	1	1	PRB	REG B DATI UNITÀ PERIFERICHE
0	0	1	0	2	DDRA	REG A DIREZ DATI
0	0	1	1	3	DDRB	REG B DIREZ DATI
0	1	0	0	4	TA LO	TIMER A REG BASSO
0	1	0	1	5	TA HI	TIMER A REG ALTO
0	1	1	0	6	TB LO	TIMER B REG BASSO
0	1	1	1	7	TB HI	TIMER B REG ALTO
1	0	0	0	8	TOD 10THS	REG DECIMI DI SECONDO
1	0	0	1	9	TOD SEC	REG SECONDI
1	0	1	0	A	TOD MIN	REG DEI MINUTI
1	0	1	1	B	TOD HR	ORE-REG AM/PM
1	1	0	0	C	SDR	REG DATI SERIALI
1	1	0	1	D	ICR	REG CONTROLL. INTERRUZ.
1	1	1	0	E	CRA	REG A DI CONTROLLO
1	1	1	1	F	CRB	REG B DI CONTROLLO

DESCRIZIONE FUNZIONALE DEL 6526

PARTI DI I/O (PRA,PRB,DDRA,DDRB)

Le parti A e B consistono ciascuna di un Registro Dati delle Unità Periferiche (PR) ad 8 bit ed un Registro della Direzione dei Dati ad 8 bit (DDR). Se un bit in DDR è posto a uno, il bit corrispondente in PR è un output; se un bit DDR è posto a 0, il bit PR corrispondente viene definito come input. Ad una LETTURA, PR riflette l'informazione presente nei pin della porta attuale (PA0-PA7,PB0-PB7) sia per il bit di input che per quello di output. La Porta A e la Porta B hanno unità di prelevamento dati passivo, che danno la compatibilità sia con CMOS che con TTL. Entrambe le parti hanno capacità di caricare due drive TTL. Oltre alle normali operazioni di I/O, PB6 e PB7 forniscono anche funzioni di output della temporizzazione.

HANDSHAKING

L'handshaking dei trasferimenti dei dati può essere eseguito usando il pin di output $\overline{PC}$ ed il pin di input $\overline{FLAG}$. $\overline{PC}$ diverrà basso per un ciclo seguendo una lettura od una scrittura della PORTA B. Questo segnale può essere usato per indicare "dati pronti" alla PORTA B o "dati accettati" dalla PORTA B. L'handshaking dei trasferimenti dei dati a 16 bit (usando sia la PORTA A che la PORTA B) è possibile sempre leggendo o scrivendo per prima cosa sulla PORTA A. $\overline{FLAG}$ è un input sensibile alla variazione negativa che può essere usato per ricevere l'output $\overline{PC}$ da un altro 6526, o come input di interruzione generale. Qualsiasi transizione negativa di $\overline{FLAG}$ porrà il bit di interruzione $\overline{FLAG}$.

REG	NOME	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	PRA	PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0
1	PRB	PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0
2	DDRA	DPA_7	DPA_6	DPA_5	DPA_4	DPA_3	DPA_2	DPA_1	DPA_0
3	DDRB	DPB_7	DPB_6	DPB_5	DPB_4	DPB_3	DPB_2	DPB_1	DPB_0

TEMPORIZZATORI (TIMER) DELL'INTERVALLO (TIMER A, TIMER B)

Ogni timer dell'intervallo consiste di un Contatore del Timer a 16 bit solo lettura e di un Latch del Timer a 16 bit solo scrittura. I dati scritti nel timer subiscono il latch nel Latch del Timer, mentre i dati letti dal timer sono i contenuti presenti nel Contatore del Timer. I timer possono essere usati in modo indipendente o connessi per operazioni estese. I vari modi timer permettono la generazione di ritardi molto lunghi, impulsi di ampiezza variabile, treni di impulsi e forme d'onda di frequenza variabile. Utilizzando l'input CNT, i timer possono contare gli impulsi esterni o misurare la frequenza, l'ampiezza dell'impulso ed i ritardi dei segnali esterni. Ogni timer ha un registro di controllo ad esso associato, che fornisce il controllo indipendente delle seguenti funzioni:

PARTENZA/FINE

Un bit di controllo permette al timer di essere fatto partire o di essere fermato dal microprocessore in qualsiasi momento.

PB ON/OFF

Un bit di controllo permette all'output del timer di apparire sulla linea di output della PORTA B (PB6 per il TIMER A e PB7 per il TIMER B). Questa funzione esclude il bit di controllo DDRB e forza la linea PB appropriata ad un output.

LEVETTA/IMPULSO

Un bit di controllo seleziona l'output applicato alla PORTA B. Ad ogni eccedenza del timer l'output può o fare da levetta o generare un impulso singolo positivo di durata di un ciclo. L'output della levetta viene posto alto tutte le volte che il timer viene fatto partire e viene posto basso da $\overline{RES}$.

IMPULSIVO/CONTINUO

Un bit di controllo seleziona ciascun modo timer. In modo impulsivo, il timer conterà alla rovescia dal valore di latch a 0, genererà un'interruzione, caricherà nuovamente il valore che ha subito il latch e poi si fermerà. In modo continuo, il timer conterà dal valore di latch a 0, genererà un'interruzione, caricherà nuovamente il valore che ha subito il latch e ripeterà la procedura continuamente.

CARICAMENTO FORZATO

Un bit dell'impulso stroboscopico permette al latch del timer di essere caricato nel contatore del timer in qualsiasi momento, che il timer stia girando oppure no.

MODO INPUT

I bit di controllo permettono alla selezione usata dal clock di decrementare il timer. Il TIMER A può contare Φ2 impulsi clock o impulsi esterni applicati al pin CNT. Il TIMER B può contare Φ2 impulsi, gli impulsi esterni CNT, gli impulsi di underflow del TIMER A o del TIMER B mentre il pin CNT è tenuto alto. Il latch del timer è caricato nel timer a qualsiasi underflow del timer, ad un caricamento forzato o seguendo una scrittura nel byte alto del precontatore mentre il timer viene fermato. Se il timer sta girando, una scrittura nel byte alto caricherà il latch del timer, ma non ricaricherà il contatore.

LETTURA (TIMER)

REG	NOME								
4	TA LO	TAL_7	TAL_6	TAL_5	TAL_4	TAL_3	TAL_2	TAL_1	TAL_0
5	TA HI	TAH_7	TAH_6	TAH_5	TAH_4	TAH_3	TAH_2	TAH_1	TAH_0
6	TB LO	TBL_7	TBL_6	TBL_5	TBL_4	TBL_3	TBL_2	TBL_1	TBL_0
7	TB HI	TBH_7	TBH_6	TBH_5	TBH_4	TBH_3	TBH_2	TBH_1	TBH_0

SCRITTURA (PRECONTATORE)

REG	NOME								
4	TA LO	PAL_7	PAL_6	PAL_5	PAL_4	PAL_3	PAL_2	PAL_1	PAL_0
5	TA HI	PAH_7	PAH_6	PAH_5	PAH_4	PAH_3	PAH_2	PAH_1	PAH_0
6	TB LO	PBL_7	PBL_6	PBL_5	PBL_4	PBL_3	PBL_2	PBL_1	PBL_0
7	TB HI	PBH_7	PBH_6	PBH_5	PBH_4	PBH_3	PBH_2	PBH_1	PBH_0

CLOCK ORA DEL GIORNO (TOD)

Il clock TOD è un timer per scopi speciali per applicazioni in tempo reale. TOD consiste di un clock 24 ore (AM/PM) con una risoluzione di 1/10mo di secondo. È organizzato in quattro registri: Decimi di secondo, Secondi, Minuti e Ore. Il flag AM/PM è nell'MSB del registro Ore per un facile esame dei bit. Ogni registro legge in formato BCD per semplificare la conversione per i video di pilotaggio, ecc. Il clock richiede un valore di input esterno a 60 Hz o 50 Hz TTL (programmabile) nel pin TOD per un controllo accurato del tempo. Oltre al controllo del tempo, viene fornito un ALLARME programmabile per la generazione di un'interruzione al momento desiderato. I registri dell'ALLARME sono localizzati agli stessi indirizzi dei registri corrispondenti TOD. L'accesso all'ALLARME è regolato dal bit del Registro di Controllo. L'ALLARME è solo scrittura; qualsiasi lettura di un indirizzo TOD leggerà il tempo senza tenere conto dello stato del bit di accesso all'ALLARME.

Una specifica sequenza di eventi deve essere seguita per una sistemazione e lettura adeguata del TOD. TOD viene fermato automaticamente tutte le volte che avviene una scrittura nel registro delle Ore. Il clock non partirà ancora fin dopo una scrittura nel registro dei decimi di secondo. Ciò assicura che TOD inizierà sempre al momento desiderato. Poichè può capitare un riporto da una fase alla seguente in qualsiasi momento in riferimento ad un'operazione di lettura, viene inclusa una funzione di latch per mantenere costanti tutte le informazioni del Tempo del Giorno durante una sequenza di lettura. Tutti i quattro registri TOD ad una lettura delle Ore subiscono il latch e rimangono così fin dopo una lettura dei decimi di secondo. Il clock TOD continua a contare quando i registri di output subiscono il latch. Se deve essere letto solo un registro, non c'è problema di riporto ed il registro può essere letto "al volo", dando per scontato che qualsiasi lettura delle Ore è seguita da una lettura dei decimi di secondo per disabilitare il latch.

LETTURA

REG	NOME								
8	TOD 10THS	0	0	0	0	T_8	T_4	T_2	T_1
9	TOD SEC	0	SH_4	SH_2	SH_1	SL_8	SL_4	SL_2	SL_1
A	TOD MIN	0	MH_4	MH_2	MH_1	ML_8	ML_4	ML_2	ML_1
B	TOD HR	PM	0	0	HH	HL_8	HL_4	HL_2	HL_1

SCRITTURA

CRB7=0 TODE)
CRB7=1 ALLARME
(STESSO FORMATO DELLA LETTURA)

PORTA SERIALE (SDR)

La porta seriale è un registro del sistema bufferizzato, con spostamento sincrono a 8 bit. Un bit di controllo seleziona il modo input o output. In modo input, il dato del pin SP viene spostato nel registro di spostamento sul fianco rialzato del segnale applicato al pin CNT. Dopo otto impulsi CNT, il dato nel registro dello spostamento viene riversato nel Registro Dati Seriali e viene generata un'interruzione. In modo output, il TIMER A viene usato per il generatore baud rate. Il dato viene spostato nel pin SP a metà dell'intervallo di underflow del TIMER A. Il massimo della baud rate possibile è $\Phi2$ diviso per 4, ma il massimo della baud rate utilizzabile sarà determinato dal caricamento della linea e dalla velocità alla quale il ricevitore risponde al dato di input. La trasmissione comincerà seguendo una scrittura nel Registro Dati Seriali (se il TIMER A sta girando ed in modo continuo). Il segnale di clock derivato dal TIMER A appare come output nel pin CNT. Il dato nel Registro Dati Seriali sarà caricato nel registro di spostamento e poi trasmesso al pin SP quando avviene un'impulso CNT. Il dato trasmesso diventa valido nel fianco di caduta di CNT e rimene valido fino al fianco di caduta seguente. Dopo otto impulsi CNT, viene generata un'interruzione per indicare che possono essere inviati ancora dati. Se il Registro Dati Seriali era stato caricato con nuove informazioni prima di questa interruzione, il dato nuovo sarà caricato automaticamente nel registro di spostamento e la trasmissione continuerà. Se il microprocessore sta un byte oltre il registro di spostamento, la trasmissione sarà continua. Se non devono essere trasmessi altri dati, dopo l'ottavo impulso CNT, CNT ritornerà alto ed SP rimarrà al valore dell'ultimo bit del dato trasmesso. Il dato SDR viene trasmesso prima in MSB ed anche i dati seriali di input dovrebbero apparire in questo formato. La capacità bidirezionale della Porta Seriale e del clock CNT permette a molte unità 6526 di essere connesse ad un bus comune di comunicazione seriale sul quale un 6526 agisce da principale, sorgente di dati e clock di spostamento mentre tutti gli altri chip 6526 agiscono da schiavi. Sia gli output CNT che quelli SP sono canali aperti che permettono tale bus comune. Il protocollo per la selezione principale/schiavo può essere trasmessa al bus seriale, o via linee di handshaking specificato.

REG	NOME								
C	SDR	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

CONTROLLO DELL'INTERRUZIONE (ICR)

Ci sono cinque sorgenti delle interruzioni sul 6526: underflow dal TIMER A, undeflow dal TIMER B, ALLARME TOD, Porta Seriale piena/vuota e FLAG. Un singolo registro fornisce le informazioni di mascheramento e di interruzione. Il Registro di Controllo dell'interruzione consiste di un registro MASCHERA di sola scrittura e di un registro DATI di sola lettura. Un'interruzione porrà il bit corrispondente nel registro DATI. Qualsiasi interruzione abilitata dal registro MASCHERA porrà il bit IR (MSB) del registro DATI e porterà basso il pin IRQ. In un sistema multi-chip il bit IR può essere interrogato per riconoscere quale chip ha generato una richiesta di interruzione. Il registro DATI interrotto viene azzerato e la linea IRQ ritorna alta seguendo una lettura del registro DATI. Poichè ogni interruzione pone un bit di interruzione senza tenere conto della MASCHERA ed ogni bit di interruzione può essere mascherato in modo selettivo per prevenire la generazione di un processore di interruzione, è possibile mischiare interruzioni chiamate con interruzioni vere. Tuttavia, chiamare il bit IR causerà l'azzeramento del registro DATI, perciò, è compito dell'utente preservare l'informazione contenuta nel registro DATI se erano presenti interruzioni chiamate.

Il registro MASCHERA fornisce un controllo conveniente di individuali bit maschera. Quando si scrive nel registro MASCHERA, se il bit 7 (SET/CLEAR) del dato scritto è ZERO, qualsiasi bit maschera scritto con un uno sarà azzerato, mentre quei bit maschera scritti con uno 0 non saranno toccati. Se il bit 7 dell dato scritto è un UNO, qualsiasi bit maschera scritto con un uno sarà posto, mentre quei bit maschera scritti con uno 0 non saranno toccati. Perchè un flag interruzione ponga un IR e generi una Richiesta di Interruzione, il bit MASCHERA corrispondente deve essere posto.

LETTURA (INT. DATI)

REG	NOME								
D	ICR	IR	0	0	FLG	SP	ALRM	TB	TA

LETTURA (INT. MASCHERA)

REG	NOME								
D	ICR	S/C	X	X	FLG	SP	ALRM	TB	TA

REGISTRI DI CONTROLLO

Ci sono due registri di controllo nel 6526, CRA e CRB. CRA è associato al TIMER A e CRB è associato al TIMER B. Il formato del registro è come segue:

CRA:

BIT	NOME	FUNZIONE
0	START	1 = TIMER A DI PARTENZA, 0 = FERMA TIMER A, questo bit è resettato automaticam. quando avviene un owerflow in modo impulsivo.
1	PB ON	1 = appare l'output del TIMER A su PB6, 0 = normale operazione PB6.
2	MODO OUT	1 = LEVETTA, 0 = IMPULSO
3	MODO RAN	1 = IMPULSIVO, 0 = CONTINUO
4	LOAD	1 = CARICAMENTO FORZATO (questo è un impulso STROBOSCOPICO, non c'è memorizzazione dei dati, il bit 4 leggerà sempre uno 0 e scrivere uno 0 non ha effetto).
5	MODO IN	1 = TIMER A conta le transizioni positive di CNT, 0 = TIMER A conta gli impulsi Φ2.
6	MODO SP	1 = output della PORTA SERIALE (sorgenti CNT del clock di spostamento), 0 = input della PORTA SERIALE (si richiede il clock di spostamento esterno).
7	TOD IN	1 = si richiede il clock a 50 Hz nel pin TOD per un tempo accurato, 0 = si richiede il clock a 60 Hz nel pin TOD per un tempo accurato.

CRB:

BIT	NOME	FUNZIONE
		(I bit CRB0-CRB4 sono uguali a CRA0-CRA4 per il TIMER B con l'eccezione che il bit 1 controlla l'output del TIMER B su PB7).
5,6	MODO IN	I bit CRB5 e CRB6 selezionano uno dei quattro modi input per il TIMER B come: CRB6 CRB5 0 0 TIMER B conta Φ2 impulsi. 0 1 TIMER B conta transizioni positive CNT. 1 0 TIMER B conta impulsi di underflow del TIMER A. 1 1 TIMER B conta impulsi di underflow del TIMER A quando CNT è alto.
7	ALARM	1 = la scrittura nei reg TOD pone ALARM, 0 = la scrittura nei reg TOD pone il clock TOD.

REG	NOME	TOD IN	MODO SP	MODO IN	CARICAM.	MODO RUN	MODO OUT	PS ON	INIZIA
E	CRA	0=60Hz 1=50Hz	0=INPUT 1=OUTPUT	0=Φ2 1=CNT	1=FORCE LOAD (STROBE)	0=CONT 1=0.5	0=PULSE 1=TOGGLE	0=PB_6OFF 1=PB_6ON	0=STOP 1=START

TA (MODO IN … INIZIA)

REG	NOME	ALLARME	IN	MODO	CARICAM.	MODO RUN	MODO OUT	PS ON	INIZIA
F	CRS	0=TOD 1= ALARM	0 1 1 1	0=Φ2 1=CNT 0=TA 1=CNT TA	1=FORCE LOAD (STROBE)	0=CONT. 1=O.S.	0=PULSE 1=TOGGLE	0=PS_7OFF 1=PS_7ON	0=STOP 1=START

TB (IN … INIZIA)

Tutti i bit non usati dei registri non vengono toccati da una scrittura e sono forzati a zero da una lettura.

MEMORIA DI ACCESSO DINAMICO CASUALE

Questa sezione tratta le caratteristiche delle RAM dinamiche del C128, che sono le RAM 4164 di 64K bit. Questa unità RAM si trova nella configurazione di 64K per 1 bit. Questa sezione contiene anche informazioni sulle RAM dinamiche 4416 usate dal controllore video 8563. Questo tipo di RAM è una RAM a 64K nella configurazione di 16K per 4 bit.

DESCRIZIONE DELLA RAM DEL SISTEMA

Il sistema C128 contiene 128K di DRAM a processore indirizzabile nella configurazione 64K per 1, organizzato in due banchi individuali da 64K. Inoltre, il sistema contiene 16K di visualizzazione locale del DRAM nel controllore video 8563, non indirizzabile direttamente dal processore.

La formazione dei banchi della RAM, descritta nei dettagli nella sezione della MMU del Capitolo 14, è controllata da parecchi registri MMU: il Registro di Configurazione, il Registro della Configurazione della RAM, i Puntatori della Pagina Zero e della Pagina Uno. In breve, il Registro di Configurazione controlla quale banco da 64K della RAM viene selezionato; il Registro della Configurazione della RAM controlla se e quanta RAM è tenuta in comune tra i banchi ed i Registri dei Puntatori dirigono di nuovo le pagine zero ed uno a qualsiasi pagina della memoria, superando l'effetto dei due Registri di Configurazione. Nel sistema, la selezione del banco RAM è ottenuta con il controllo del CAS che ha subito il gate.

CARATTERISTICHE FISICHE

Questa sezione tratta alcune delle caratteristiche della RAM di 64K per 1 bit e della RAM 16K per 4 bit che sono usate nel sistema C128. Una tabella del pinout ed una figura vengono date sia per il pacchetto 4164 che per quello 4416 (consultate la Tabella 17-15 e 17-16 e le Figure 17-23 e 17-24).

PIN	NOME	DESCRIZIONE
1	NC	Non collegato
2	D_{in}	Dato Inserito
3	/WE	Abilitazione Scrittura (Attiva Bassa)
4	/RAS	Impulso Stroboscopico dell'Indirizzo di Riga (Attivo Basso)
5	A_0	Bit 0 dell'Indirizzo
6	A_2	Bit 2 dell'Indirizzo
7	A_1	Bit 1 dell'Indirizzo
8	V_{CC}	Alimentazione +5 Vdc
9	A_7	Bit 7 dell'Indirizzo
10	A_5	Bit 5 dell'Indirizzo
11	A_4	Bit 4 dell'Indirizzo
12	A_3	Bit 3 dell'Indirizzo
13	A_6	Bit 6 dell'Indirizzo
14	D_{out}	Dato Disinserito/Fuori
15	/CAS	Impulso Stroboscopico dell'Indirizzo di Colonna (Attivo Basso)
16	V_{SS}	Terra dell'Alimentazione

Tabella 17-15. Pinout 4164

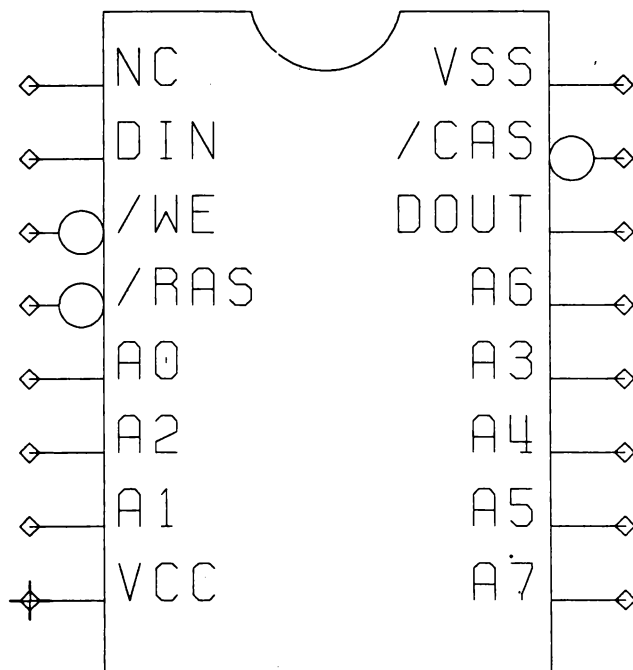



Figura 17-23. Il Chip 4164

PIN	NOME	DESCRIZIONE
1	/OE	Abilitazione dell'Output (Attiva Bassa)
2	D_1	Bit 1 del Dato
3	D_2	Bit 2 del Dato
4	/WE	Abilitazione della Scrittura (Attiva Bassa)
5	/RAS	Impulso Stroboscopico dell'Indirizzo di Riga (Attivo Basso)
6	A_6	Bit 6 dell'Indirizzo
7	A_5	Bit 5 dell'Indirizzo
8	A_4	Bit 4 dell'Indirizzo
9	V_{dd}	Alimentazione +5 Vdc
10	A_7	Bit 7 dell'Indirizzo
11	A_3	Bit 3 dell'Indirizzo
12	A_2	Bit 2 dell'Indirizzo
13	A_1	Bit 1 dell'Indirizzo
14	A_0	Bit 0 dell'Indirizzo
15	D_3	Bit 3 del Dato
16	/CAS	Impulso Stroboscopico dell'Indirizzo di Colonna (Attivo Basso)
17	D_4	Bit 4 del Dato
18	V_{ss}	Terra dell'Alimentazione

Tabella 17-16. Pinout 4416

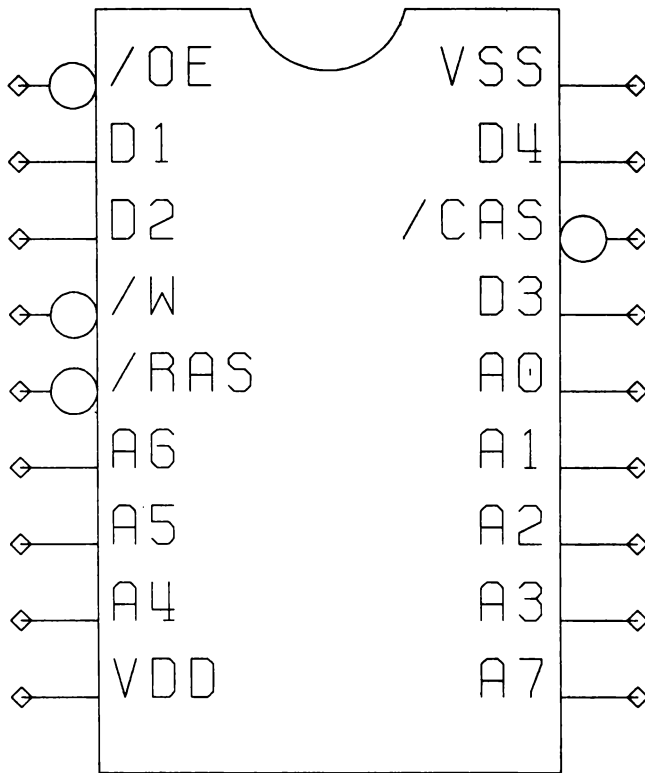



Figura 17-24. Il Chip 4464

MEMORIA DI SOLA LETTURA (ROM)

Questa sezione descrive la ROM del sistema C128, sia da un punto di vista logico che da un punto di vista hardware. Menziona aspetti della struttura a banchi della ROM, la gestione del Kernal e del BASIC e spiega le specifiche fisiche delle stesse unità ROM.

DESCRIZIONE DELLA ROM DEL SISTEMA

In modo C64, il sistema operativo risiede in 16 K della ROM, che include approssimativamente 8K per il Kernal ed 8K per il BASIC. In modo C128, il sistema operativo risiede in 48K della ROM ed include il Kernal avanzato e le funzioni BASIC. Il Kernal, per definizione, è il sistema operativo generale del computer, con punti di entrata specificati nelle subroutine utilizzabili per facilitare l'uso trasparente dell'aggiornamento della ROM da parte di programmi a più alto livello. C'è anche una ROM caratteri che risiede nel Bus Condiviso, condiviso dal chip VIC e dal processore. Questa ROM è una 4016 da 8K per 8, NMOS ROM. La ROM OS C64 è così cablata da apparire come fosse formata da due tronconi di ROM non contigua, che copia l'attuale mappa della memoria della ROM C64. Viene incluso un collaudo per gestire il sistema della ROM o come 4 ROM di 16K per 8 o come due ROM di 32K per 8. Tutte le funzioni interne delle ROM del C128 saranno la variazione dei 32K per 8.

FORMAZIONE DEI BANCHI DELLA ROM

Fate riferimento alla mappa del registro della MMU, Figura 14-4 del Capitolo 14. Notate che il Registro della Configurazione (CR) controlla il tipo di ROM o di RAM dato in una determinata locazione di indirizzo. A seconda dei contenuti di CR, la ROM può essere abilitata o disabilitata per ottenere la configurazione più utile per l'applicazione che si ha sotto mano. La ROM viene abilitata in tre aree della memoria in modo C128, ognuna consistente di 16K di spazio dell'indirizzo. La ROM più bassa può essere definita come RAM o ROM del Sistema, le due ROM più alte possono essere ROM del Sistema, Funzioni ROM, Cartucce ROM o RAM. In modo C64, si applicano le regole della mappa della memoria C64, che sono primitive se comparate con quelle usate in modo C128. La ROM C64 è formata da banchi come fosse in due sezioni da 8K, BASIC e Kernal, secondo la porta della pagina zero e la cartuccia al suo posto in quel momento. Non può avvenire una libera formazione di banchi quando una cartuccia è al suo posto. In C128,

se un indirizzo cade nell'intervallo di una ROM abilitata, MMU comunicherà lo stato della ROM al decodificatore PLA attraverso le linee di stato della memoria. Essenzialmente, MMU guarda nel Registro di Configurazione quale ROM o RAM è posta. Consultate il Capitolo 14. Il modo in cui lo schema dei banchi è implementato permette fino a 32K di ROM interna, divisibile in banchi per l'utilizzo in programmi come le Applicazioni dei Tasti Funzioni e sopporterà 32K di ROM interna divisibile in banchi. Varie combinazioni della ROM sono possibili e possono essere notate studiando le configurazioni del Registro di Configurazione. I tipi di ROM 23128 (16K per 8) e 23256 (32K per 8) sono usati dal sistema.

SPECIFICHE DELLA TEMPORIZZAZIONE

ROM INTERNE

Questa sezione specifica i parametri della temporizzazione sia per la Memoria Solo Lettura 23128 che per quella 27256. Queste spec di temporizzazione si adattano alle ROM interne e per le ROM esterne girano a 1 MHz. Le ROM esterne girano a 2 MHz, consultate la Tabella 17-18.

PARAMETRO	SIMBOLO	MIN	MAX	UNITÀ
Indirizzo Valido ritardo in Output	T_{ACC}	300	-	ns
Abilitazione del Chip per un ritardo in Output	T_{CE}	300	-	ns
Abilitazione dell'Output per un ritardo in Output	T_{OE}	120	-	ns

Tabella 17-17. Temporizzazione della ROM Interna

INDIRIZZO

ABILITAZIONE DEL CHIP

ABILITAZIONE DELL'OUTPUT

DATO

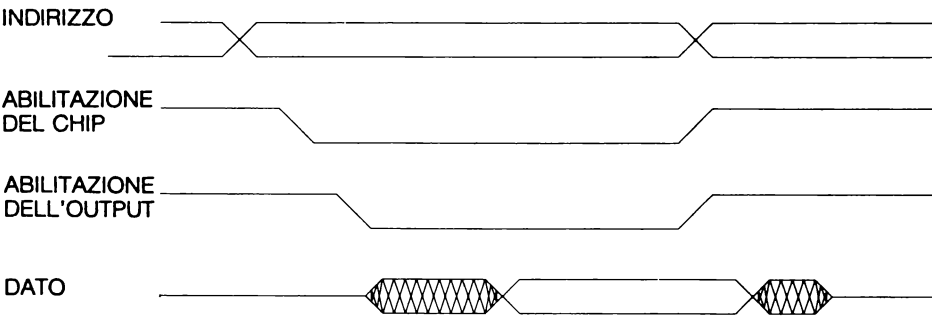



Figura 17-25. Diagramma della temporizzazione della ROM

ROM ESTERNE

Tutte le ROM esterne del modo C64 e molte ROM esterne del modo C128 possono essere del tipo menzionato prima. Qualsiasi ROM esterna che gira a 2 MHz deve essere più veloce, come specificato nella Tabella 17-18.

PARAMETRO	SIMBOLO	MIN	MAX	UNITÀ
Indirizzo Valido per un ritardo Output	T_{ACC}	250	-	ns
Abilitazione del Chip per un ritardo in Output	T_{CE}	200	-	ns
Abilitazione dell'Output per un ritardo in Output	T_{OE}	100	-	ns

Tabella 17-18. Temporizzazione della ROM esterna a 2MHz

LA ROM 23128

PIN	NOME	DESCRIZIONE
1	V_{PP}	Voltaggio di Programmazione
2	A_{12}	Bit 12 dell'Indirizzo (A_{13} nella ROM OS C64)
3	A_7	Bit 7 dell'Indirizzo
4	A_6	Bit 6 dell'Indirizzo
5	A_5	Bit 5 dell'Indirizzo
6	A_4	Bit 4 dell'Indirizzo
7	A_3	Bit 3 dell'Indirizzo
8	A_2	Bit 2 dell'Indirizzo
9	A_1	Bit 1 dell'Indirizzo
10	A_0	Bit 0 dell'Indirizzo
11	D_0	Bit 0 del Dato
12	D_1	Bit 1 del Dato
13	D_2	Bit 2 del Dato
14	GND	Terra dell'Alimentazione
15	D_3	Bit 3 del Dato
16	D_4	Bit 4 del Dato
17	D_5	Bit 5 del Dato
18	D_6	Bit 6 del Dato
19	D_7	Bit 7 del Dato
20	/CE	Abilitazione del Chip (Attiva Bassa)
21	A_{10}	Bit 10 dell'Indirizzo
22	/OE	Abilitazione dell'Output (Attiva Bassa)
23	A_{11}	Bit 11 dell'Indirizzo
24	A_9	Bit 9 dell'Indirizzo
25	A_8	Bit 8 dell'Indirizzo
26	A_{13}	Bit 13 dell'Indirizzo
27	/PGM	Abilitazione del Programma (Attiva Bassa)
28	V_{CC}	Alimentazione +5 Vdc

Tabella 17-19. Pinout della ROM 23128

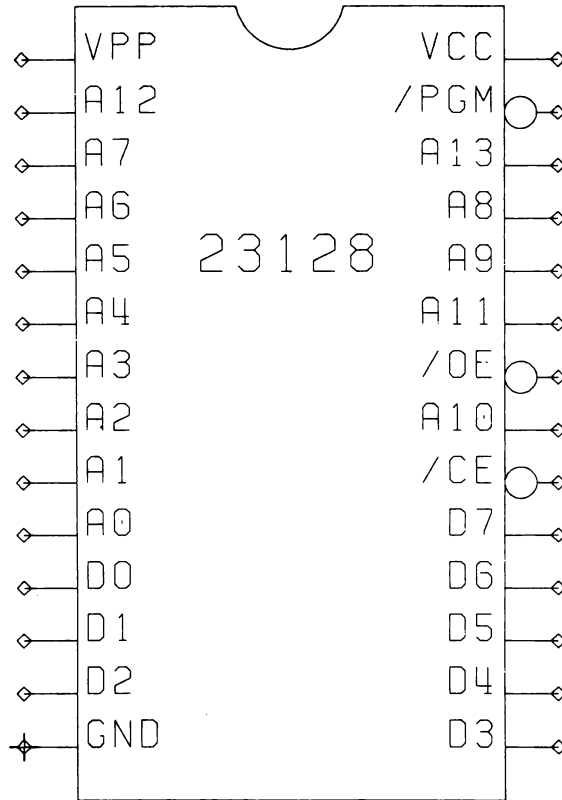



Figura 17-26. Chip della ROM 23128 (BASIC, Kernal, Funzioni Editor ed Esterne delle ROM)

LA ROM 23256

PIN	NOME	DESCRIZIONE
1	V_{pp}	Voltaggio di Programmazione
2	A_{12}	Bit 12 dell'Indirizzo
3	A_7	Bit 7 dell'Indirizzo
4	A_6	Bit 6 dell'Indirizzo
5	A_5	Bit 5 dell'Indirizzo
6	A_4	Bit 4 dell'Indirizzo
7	A_3	Bit 3 dell'Indirizzo
8	A_2	Bit 2 dell'Indirizzo
9	A_1	Bit 1 dell'Indirizzo
10	A_0	Bit 0 dell'Indirizzo
11	D_0	Bit 0 del Dato
12	D_1	Bit 1 del Dato
13	D_2	Bit 2 del Dato
14	GND	Terra dell'Alimentazione
15	D_3	Bit 3 del Dato
16	D_4	Bit 4 del Dato
17	D_5	Bit 5 del Dato
18	D_6	Bit 6 del Dato
19	D_7	Bit 7 del Dato
20	/CE-/PGM	Abilitazione del Chip - Abilitazione del Programma (Attiva Bassa)
21	A_{10}	Bit 10 dell'Indirizzo
22	/OE	Abilitazione dell'Output (Attiva Bassa)
23	A_{11}	Bit 11 dell'Indirizzo
24	A_9	Bit 9 dell'Indirizzo
25	A_8	Bit 8 dell'Indirizzo
26	A_{13}	Bit 13 dell'Indirizzo
27	A_{14}	Bit 14 dell'Indirizzo
28	V_{cc}	Alimentazione +5 Vdc

Tabella 17-20. Pinout della ROM 23256 (ROM Funzioni Esterne o Interne)

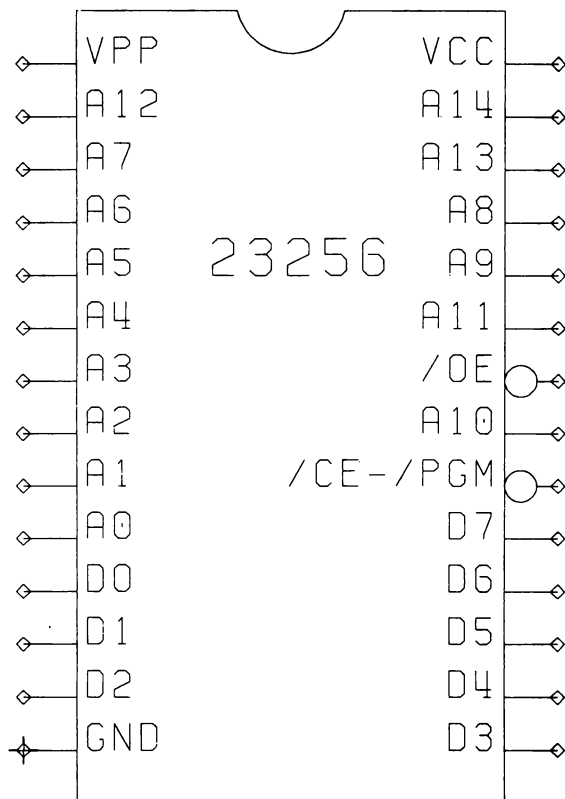



Figura 17-27. Il Chip della ROM 23256

IL BUS SERIALE

Il Bus Seriale C128 è una versione migliorata del bus seriale C64. Questo bus usa il tipo di indirizzamento IEEE-488, mantenendo così la compatibilità software tra le unità seriali del mercato dei consumatori e le unità parallele CBM IEEE-488. Il C128 migliora questo bus permettendo la comunicazione a velocità molto più alte con unità periferiche progettate in modo speciale, il più importante è il disk drive, mentre mantiene ancora la compatibilità con le unità periferiche più vecchie e più lente usate dal C64. Questa sezione descrive l'hardware ed alcuni degli aspetti di software di entrambi gli schemi seriali di trasmissione vecchio e nuovo.

OPERAZIONI DEL BUS

Ci sono tre operazioni basilari del bus che avvengono nel bus seriale, sia in modo veloce che lento. La prima è chiamata **Control.** Il C128 è il controllore nella maggior parte delle circostanze. Il controllore del bus è sempre l'unità che inizializza il protocollo del bus, richiedendo alle unità periferiche di fare una delle altre due operazioni seriali, o **Talk** o **Listen.**

Tutte le unità del bus seriale possono ascoltare. Un'unità di ascolto è un'unità a cui è stato ordinato da parte del controllore di ricevere i dati. Alcune unità, come i disk drive, possono parlare. Un'unità parlante invia i dati al controllore. Sia l'hardware che il software pilotano questo protocollo del bus.

SEGNALI DEL BUS

Il bus seriale Commodore è composto dai seguenti segnali:

- SRQ (pin 1): Questo segnale è chiamato **Richiesta di Servizio**. Il bus seriale lento non usa questa linea, la linea del clock veloce bidirezionale. (Non usata in Modo C64).
- GND (pin 2): Terra della Scheda Madre
- ATN (pin 3): Questo segnale è chiamato **Attenzione.** È usato per indirizzare un'unità al bus. Quando si richiede ad un'unità di parlare o di ascoltare, il controllore porta questo segnale basso, creando una specie di interruzione di tutti i bus delle unità seriali. Poi emette un indirizzo che selezionerà un'unità del bus. È compito del controllore superare il tempo se un'unità del bus non risponde entro un periodo ragionevole di tempo.
- CLK (pin 4): Questo è il clock seriale lento. È usato da unità seriali lente, che sono tempificate dal software, per tempificare i dati trasmessi al bus seriale.

- DATA (pin 5): Questa è la linea dei dati seriali. È usata sia dalle unità seriali lente che da quelle veloci per trasmettere i dati in sincronizzazione con il segnale del clock.
- RESET (pin 6): Questa è la linea di reset, usata per resettare tutte le unità periferiche quando l'elaboratore ospite resetta.

BUS SERIALE VELOCE

PROTOCOLLO VELOCE

Per funzionare come parlatore veloce, il sistema deve indirizzare un'unità veloce, come il disk drive 1571. Quando indirizza qualsiasi unità, il C128 invia un byte veloce, facendo da levetta alla linea SQR per 8 volte, mentre la linea ATN è bassa. Se l'unità è un'unità veloce, registrerà il fatto che un controllore veloce ha avuto accesso ad essa e risponderà con un'accettazione veloce. Se l'unità è un'unità lenta, non viene data risposta ed allora il C128 assume che sta parlando con un'unità lenta. Lo stato della velocità del drive viene mantenuto finchè viene chiesto all'unità di non parlare o di non ascoltare, se capita un errore, o se capita un reset del sistema.

HARDWARE VELOCE

Il bus seriale veloce, per aumentare la sua velocità, usa diversi hardware rispetto a quello del bus seriale lento. Il bus seriale lento usa parecchie linee della porta 6526 per pilotare ATN, CLK e DATA. Così, la temporizzazione dei trasferimenti dei dati deve essere pilotata dal software. Il metodo seriale veloce è di usare la linea della porta seriale di un 6526 (CIA-1) per trasferire realmente i dati seriali. Questo aumenta molto la velocità di trasferimento, al punto in cui il trasferimento diventa limitato più dall'overhead del software che da qualsiasi altra cosa. La velocità reale del trasferimento viene posta dal temporizzatore 6526. I 6526 correnti hanno un valore minimo del temporizzatore seriale di 4, sebbene nell'uso reale questo valore sia più vicino a 6, a causa del caricamento. Qualsiasi alimentazione del 6526 renderebbe possibile un trasferimento più veloce dei dati.

Questo schema potrebbe interferire con le trasmissioni seriali lente, poichè la linea DATA viene condivisa da entrambi gli schemi. Così, esistono i circuiti elettronici che isoleranno i driver veloci seriali dal bus seriale lento. Porre FSDIR in modo input è sufficiente a rimuovere qualsiasi interazione veloce seriale possibile con il bus seriale lento, diversa dal caricamento di un'ulteriore unità, che non è un problema a velocità lente del bus seriale.

Per assicurare la compatibilità con il C64, tuttavia, il bus seriale lento non può interferire con i driver veloci, poichè questi driver sono in comune all'Ingresso dell'Utente ed un'unità ingresso dell'utente potrebbe presumibilmente farne uso. Una volta che viene posto il modo C64, la direzione di input dei circuiti dell'interfaccia è disabilitata. Così, in modo C64, il bit FSDIR deve essere posto in input

per rimuovere l'interferenza del veloce sul lento, ma l'interferenza del lento sul veloce viene rimossa automaticamente chiamando il modo C64. Non c'è modo di disabilitare l'interferenza del lento sul veloce in modo C128 (almeno non simultaneamente con l'eliminazione dell'interferenza del veloce sul lento).

IL BUS DI ESPANSIONE

Il Bus di Espansione C128 è compatibile con il Bus di Espansione C64, mentre allo stesso tempo permette capacità estese in modo C128.

AGGIUNTA DELLA CARTUCCIA

Il C128 può usare cartucce più grandi e più sofisticate rispetto al C64. Una delle maggiori ragioni di questo è il nuovo schema dei banchi usato in C128 per le cartucce esterne. Il C64 usa due linee di controllo hardware, /EXROM e /GAME, per controllare la formazione dei banchi fuori dai mezzi interni e dentro le cartucce. Il C128 usa un metodo software di interrogazione ciclica, dove al momento dell'accensione esso interroga la cartuccia, secondo un determinato protocollo, per determinare se esiste tale cartuccia, e se è così, quale priorità software ha. Poichè il C128 è sempre libero di porre i banchi tra le cartucce e le ROM incorporate, un'applicazione esterna può avvantaggiarsi delle routine interne e diventare naturalmente una parte estesa del C128, opposto al fatto di diventare un'applicazione di riserva. Consultate il Capitolo 14 per informazioni sulla sequenza ROM di Cartucce Auto Inizianti.

L'eliminazione di /EXROM e /GAME come linee di controllo hardware per l'identificazione delle cartucce (in modo C128) ha liberato entrambe queste linee per un funzionamento esteso. Entrambe le linee appaiono come bit nel Registro del Modo della Configurazione MMU e sono entrambe porte di input e output. Nessuna ha una funzione determinata diversa dalla funzione generale di espansione della cartuccia e si prestano ad agire da linee di formazione di banchi soggetti al latch o linee di lettura di input. Naturalmente, nessuna può essere posta al momento dell'accensione del C128 oppure sarà posto automaticamente il modo C64.

CAPACITÀ DMA

Il bus di espansione C128 sopporta i DMA in forma simile a quella del C64. Un DMA C64 è ottenuto spingendo il pin /DMA basso nel bus di espansione. Subito dopo che ciò è avvenuto, le linee RDY e AEC del processore divengono basse. Questo può chiudere nettamente il processore, ma può causare anche dei problemi, dipende da ciò che il processore sta facendo in quel momento. L'input RDY di un processore in serie 8502, quando è portato basso, fermerà il processore del ciclo Φ1 successivo, lasciando che le linee di indirizzo del processore riflettano l'indirizzo corrente che viene preso. Tuttavia, se il processore è in un ciclo di scrittura quando RDY è portato basso, ignorerà RDY fino al ciclo di lettura successivo. Così, in C64, un input /DMA che avviene durante un ciclo di scrittura renderà a tre stati l'indirizzo del processore ed il bus dei dati, ma non li fermerà fino a tre cicli dopo, quando avviene il ciclo di lettura successivo. I cicli di scrittura che seguono l'input /DMA non scrivono in realtà, causando il danneggiamento della memoria e spesso il danneggiamento totale del processore quando la linea /DMA viene rilasciata. Qualsiasi input /DMA durante Φ2 è un potenziale DMA fatale.

Se viene posto un /DMA adeguato, il C64 diviene tri-stato e si chiude, permettendo alla sorgente DMA un pieno accesso al bus del processore. Tale sorgente DMA deve inviare al monitor gli output Φ2 e BA, poichè deve essere tri-stato quando VIC è nel bus e deve subire completamente il DMA quando viene chiamato un DMA VIC. Il chip VIC ha sempre la priorità DMA più alta. Quando è sul bus, la sorgente DMA accede alla RAM, ROM e I/O dello schema C64. Una chiusura adeguata del DMA viene di solito ottenuta per mezzo di alcuni handshaking software C64 con la sorgente DMA.

Il sistema C128 usa uno schema DMA simile. Quando l'input /DMA diventa basso, l'input RDY verso l'8502, l'input AEC verso l'8502 e l'input /BUSRQST verso lo Z80 diventano bassi immediatamente. Inoltre, il segnale AEC che ha subito il gate, GAEC, diventa basso fecendo andare l'MMU subito nel suo MODO di CICLO VIC e rendendo tri-stati il buffer di Uscita Dati dello Z80. Il DMA fà cambiare direzione all'Indirizzo verso il buffer dell'Indirizzo Condiviso e abilita l'Indirizzo Tradotto verso il buffer dell'Indirizzo, dando alla sorgente esterna DMA pieno accesso al bus dell'indirizzo del processore. PLA sta ancora controllando AEC che non ha subito il gate e come tale permetterà l'accesso alle unità di I/O, RAM e ROM. Ci può non essere accesso alla MMU; così per formare la mappa della memoria del C128 la mappa della memoria deve essere predisposta prima di subire il DMA. Per il modo C64, la formazione della mappa della memoria viene effettuata dalle linee delle porte del processore 8502 e da /EXROM e /GAME esterni. Poichè le porte dell'8502 saranno inaccessibili per una fonte DMA, si possono effettuare solo i cambiamenti della mappa C64 basati su /EXROM e /GAME durante un DMA. Questo è uguale per un'unità C64. Tutte le sorgenti DMA, come per il C64, devono adattarsi al VIC durante Φ0 o BA basso. Il C128 può eseguire un DMA distruttivo facilmente come nel C64. Per usare i DMA, la sorgente DMA deve molto probabilmente cooperare con un programma in modo C128, permettendogli di eseguire l'handshake con una sorgente DMA per compiere i DMA in modo non distruttivo.

PINOUT DEL BUS DI ESPANSIONE

PIN	NOME	DESCRIZIONE
1	GND	Terra del Sistema
2	+5V	V_{cc} del Sistema
3	+5V	V_{cc} del Sistema
4	/IRQ	Richiesta di Interruzione
5	R/W	Segnale di Lettura Scrittura del Sistema
6	DCLOCK	Clock Dot del Video a 8.18 MHz
7	I/O_1	Selezione I/O del Chip: $DE00-$DEFF, Attiva Bassa
8	/GAME	Esplorata per la Configur. della Mappa della Memoria
9	/EXROM	Esplorata per la Configur. della Mappa della Memoria
10	I/O_2	Selezione I/O del Chip: $DF00-$DFFF, Attiva Bassa
11	$/ROM_L$	Selezione del Chip della ROM Esterna, $8000-$BFFF in Modo C128 ($8000-$9FFF in Modo C64)
12	BA	Output del Bus Disponibile
13	/DMA	Input di Accesso Diretto alla Memoria (consultate le precauzioni delle capacità DMA a pag. precedente)
14	D_7	Bit 7 del Dato
15	D_6	Bit 6 del Dato
16	D_5	Bit 5 del Dato
17	D_4	Bit 4 del Dato
18	D_3	Bit 3 del Dato
19	D_2	Bit 2 del Dato
20	D_1	Bit 1 del Dato
21	D_0	Bit 0 del Dato
22	GND	Terra del Sistema
A	GND	Terra del Sistema
B	$/ROM_H$	Selezione del Chip della ROM esterna, $C000-$FFFF in Modo C128 ($C000-$FFFF in modo C64)
C	/RESET	Segnale di Reset del Sistema
D	/NMI	Richiesta di Interruzione Non Mascherabile
E	1MHZ	Clock del Sistema Φ_0 a 1 MHz
F	TA_{15}	Bit 15 dell'Indirizzo Tradotto
H	TA_{14}	Bit 14 dell'Indirizzo Tradotto
J	TA_{13}	Bit 13 dell'Indirizzo Tradotto
K	TA_{12}	Bit 12 dell'Indirizzo Tradotto
L	TA_{11}	Bit 11 dell'Indirizzo Tradotto
M	TA_{10}	Bit 10 dell'Indirizzo Tradotto
N	TA_9	Bit 9 dell'Indirizzo Tradotto
P	TA_8	Bit 8 dell'Indirizzo Tradotto
R	SA_7	Bit 7 dell'Indirizzo Comune/Condiviso
S	SA_6	Bit 6 dell'Indirizzo Comune
T	SA_5	Bit 5 dell'Indirizzo Comune
U	SA_4	Bit 4 dell'Indirizzo Comune
V	SA3	Bit 3 dell'Indirizzo Comune
W	SA_2	Bit 2 dell'Indirizzo Comune
X	SA_1	Bit 1 dell'Indirizzo Comune
Y	SA_0	Bit 0 dell'Indirizzo Comune
Z	GND	Terra del Sistema

Tabella 17-21. Pinout del bus di espansione

L'INTERFACCIA DEL VIDEO

L'interfaccia video VIC dell'hardware C128 permette la connessione di un televisore standard commerciale NTSC o PAL e/o un monitor a colori. Il monitor può accettare o un segnale di video composto o segnali separati del colore e luminosità/sincronia oltre ad un segnale audio. Questo output è molto simile all'output delle unità C64 revisione successiva.

Il C128 fornisce anche l'interfacciamento di un video ad 80 colonne. Il video ad 80 colonne disponibile è RGBI e monocromatico, in grado di interfacciare con la maggior parte dei monitor NTSC- o PAL-compatibili di TIPO RGBI I e con la maggior parte dei monitor compatibili con NTSC o PAL monocromatici ad 80 colonne.

INTERFACCIA VIDEO VIC

I seguenti paragrafi specificano l'interfaccia video VIC per il video a 40 colonne in 16 colori. Il segnale VIC è disponibile per i valori analogici del connettore video e per i valori RF per l'output RF.

SPECIFICHE DEL MODULATORE

Il modulatore fornisce un segnale RF di tipo trasmissione che riporta i segnali di video ed audio composto VIC. Il modulatore NTSC è variabile tra i canali 3 e 4 per aiutare la minimalizzazione dell'interferenza locale della trasmissione. Il segnale generato dal modulatore RF si adegua alle regole FCC concernenti la Classe B FCC, le unità di interfaccia della TV. L'output RF è accessibile attraverso il jack standard di tipo RCA fono/video.

OUTPUT DEL MONITOR

L'output del video VIC fornisce i segnali mostrati nella Tabella 17-22.

SEGNALE	VALORE	IMPEDENZA	OFFSET DC
Lum/Sinc	1 Vp-p	75 Ω	0.5 V
Colore	1 Vp-p	75 Ω	0.5 V
Composto	1 Vp-p	75 Ω	0.5 V
Audio	1 Vp-p	1K Ω	

Tabella 17-22. Segnali di output del Video VIC

PINOUT DEL CONNETTORE VIDEO

Il connettore del video VIC esiste fisicamente come connettore DIN a 8 pin. Fornisce i segnali mostrati nella Tabella 17-23.

PIN	SEGNALE
1	Luminosità/Sincron.
2	Terra
3	Audio Out
4	Composto
5	Audio In
6	Colore
7	N.C.
8	N.C.

Tabella 17-23. Segnali del connettore Video

INTERFACCIA VIDEO 8563

I seguenti paragrafi specificano l'interfaccia video 8563 per il video ad 80 colonne a 16 colori. Il segnale 8563 è disponibile con valori digitali per l'RGBI e con un analogico derivato da tre valori per il video composto in bianco e nero.

OUTPUT DEL MONITOR

La Tabella 17-24 mostra i segnali forniti dall'output dell'8563.

SEGNALE	VALORE	IMPEDENZA
Rosso	TTL	TTL
Verde	TTL	TTL
Blu	TTL	TTL
Intensità	TTL	TTL
Sincron. O	TTL	TTL
Sincron. V	TTL	TTL
Composto		75 Ω
Intensità Piena	2.0 V	
Mezza Intensità	1.5 V	
Sincron.	0.5 V	

Tabella 17-24. Segnali di output dell'8563

PINOUT DEL CONNETTORE VIDEO

Il connettore video 8563 è un connettore D9 stile IBM, che fornisce i segnali mostrati nella Tabella 17-25.

PIN	SEGNALE
1	Terra
2	Terra
3	Rosso
4	Verde
5	Blu
6	Intensità
7	Monocromatico (non standard)
8	Sincronizzazione orizzontale
9	Sincronizzazione verticale

Tabella 17-25. Pinout del connettore video 8563

LA TASTIERA

La tastiera C128 è una tastiera avanzata successiva a quella standard del C64, mentre mantiene ancora la piena compatibilità. Ha parecchi tasti extra che sono usati in modo C128, ma non in modo C64. Ha come funzione una tastiera numerica, un tasto **HELP,** tasti con funzioni estese, un vero tasto **CAPS LOCK** ed un tasto di variabilità **40/80** colonne, tutti sono inviati con impulso stroboscopico dal chip VIC o legati ad un 8502 specifico o a linee di I/O della MMU.

PINOUT DEL CONNETTORE

La tastiera del C128 è progettata per essere connessa da un connettore singolo in programma di linea interno a 12- e 13-pin per l'unità con una tastiera incorporata. La Tabella 17-26 illustra entrambe le connessioni.

D-TYPE	SEGNALE
1	Terra
2	Tasto
3	Ripristino
4	+5V
5	Riga 3
6	Riga 6
7	Riga 5
8	Riga 4
9	Riga 7
10	Riga 2
11	Riga 1
12	Riga 0
13	Colonna 0
14	Colonna 6
15	Colonna 5
16	Colonna 4
17	Colonna 3
18	Colonna 2
19	Colonna 1
20	Colonna 7
21	K_0
22	K_1
23	K_2
24	40/80
25	Alpha Lock

Tabella 17-26. Pinout del connettore della tastiera

TABELLA DELLA TASTIERA DEL C128

	C0 PIN13	C1 PIN19	C2 PIN18	C3 PIN17	C4 PIN16	C5 PIN15	C6 PIN14	C7 PIN20	K0 PIN21	K1 PIN22	K2 PIN23
R0 PIN12	INS DEL	# 3	% 5	, 7	) 9	+	1b	? 1	HELP	ESC	ALT
R1 PIN10	RET	W	R	Y	I	P	•	←	8	+	0
R2 PIN10	↑ ↓	A	D	G	J	L	] ;	CTRL	5	-	.
R3 PIN5	F8 F7	$ 4	& 6	(8	0	-	CLR HOM	" 2	TAB	LINE FEED	↑
R4 PIN8	F2 F1	Z	C	B	M	> .	RIGHT SHIFT	SPACE BAR	2	ENTER	↓
R5 PIN7	F4 F3	S	F	H	K	[:	=		4	6	←
R6 PIN6	F6 F5	E	T	U	O	@	π	Q	7	9	→
R7 PIN9	↑ →	LEFT SHIFT	X	V	N	< ,	? /	RUN STOP	1	3	NO SCRL

GN PIN-1

SHIFT LOCK — BLOCCAGGIO

NMI PIN3 ← RESTR

40/80 PIN24 ← 40/80 DSPLY — BLOCCAGGIO

P6510 PIN25 ← CAPS LOCK — BLOCCAGGIO

NOTA: I pin da R0 a R7 appartengono ai valori di riga della tastiera per la SCANSIONE della tastiera. Questi pin corrispondono ai bit da 0 a 7 di locazione 56321 ($DC01). I pin da C0 a C7 sono i valori di colonna della tastiera, che corrispondono ai bit da 0 a 7 di locazione 56320 ($DC00). I pin da K0 a K2 appartengono al bit del registro di controllo della tastiera C128, da 0 a 2 della locazione 53295 ($D02F).

APPENDICI

APPENDICE A

MESSAGGI DI ERRORE IN LINGUAGGIO BASIC

I seguenti messaggi di errore sono visualizzati dal BASIC. I messaggi di errore possono essere visualizzati anche con l'uso della funzione ERR$(). I numeri di errore qui sotto si riferiscono al numero assegnato all'errore per l'uso della funzione ERR$().

ERRORE #	NOME ERRORE	DESCRIZIONE
1	TOO MANY FILES	C'è un limite di 10 file APERTI per volta.
2	FILE OPEN	È stato fatto un tentativo di aprire un file usando il numero di un file già aperto.
3	FILE NOT OPEN	Il numero di file specificato in una istruzione I/O deve essere aperto prima dell'uso.
4	FILE NOT FOUND	Non esiste file con quel nome (dischetto) o è stato letto un segno di fine nastro (nastro).
5	DEVICE NOT PRESENT	L'unità di I/O richiesta non è disponibile o i buffer sono disallocati (cassetta). Controllate per verifica di connessione di unità e accens.
6	NOT INPUT FILE	Si è tentato di GET o INPUT un dato dal file specificato come solo output.
7	NOT OUTPUT FILE	Si è tentato di inviare i dati ad un file specificato come solo input.
8	MISSING FILE NAME	Manca nome del file nel comando.
9	ILLEGAL DEVICE NUMBER	Si è tentato di usare un'unità in modo sbagliato (SALVA allo schermo, ecc).
10	NEXT WITHOUT FOR	Entrambi i cicli sono intercalati non correttamente o c'è un nome variabile in una istruzione NEXT che non corrisponde a quella in FOR.
11	SYNTAX	Istruzione non riconosciuta da BASIC. A causa di una parentesi di troppo o mancante una parola chiave sbagliata,ecc.

ERRORE #	NOME ERRORE	DESCRIZIONE
12	RETURN WITHOUT GOSUB	Istruzione RETURN incontrata quando non era attiva l'istruzione GOSUB.
13	OUT OF DATA	Incontrata istruzione READ senza nessun dato da leggere (READ).
14	ILLEGAL QUANTITY	Numero usato come argomento di una funzione o istruzione è oltre l'intervallo permesso.
15	OVERFLOW	Il risultato di un calcolo è maggiore del max numero permesso (1.701411834E + 38).
16	OUT OF MEMORY	Non c'è più spazio per il codice del programma e/o variabili del program, o ci sono troppe istruzioni DO, FOR o GOSUB intercalate.
17	UNDEF'D STATEMENT	Un numero di linea a cui ci si è riferiti non esiste nel programma.
18	BAD SUBSCRIPT	Il programma ha tentato di far riferimento ad un elemento di una matrice fuori dall'intervallo specificato dalla istruzione DIM.
19	REDIM'D ARRAY	Una matrice può essere DIMensionata solo una volta.
20	DIVISION BY ZERO	La divisione per zero non è permessa.
21	ILLEGAL DIRECT	Istruzioni INPUT, GET, INPUT#, GET# e GETKEY sono solo permesse dentro un programma.
22	TYPE MISMATCH	Questo errore avviene quando un valore numerico è assegnato ad una variabile stringa o viceversa.
23	STRING TOO LONG	Una stringa può contenere fino a 255 caratteri.
24	FILE DATA	Errata lettura di dati da un file di nastro o dischetto.
25	FORMULA TOO COMPLEX	Il computer non è stato in grado di valutare questa espressione. Semplificala (rompila in due o usa meno parentesi).
26	CAN'T CONTINUE	Il comando CONT non funziona se il programma non è fatto girare, se c'era un errore o se una linea è stata editata.
27	UNDEF'D FUNCTION	Ci si è riferiti ad una funzione definita dall'utente che non fu mai definita.
28	VERIFY	Il programma su nastro o dischetto non corrisponde al programma in memoria.

ERRORE #	NOME ERRORE	DESCRIZIONE
29	LOAD	Problema di caricamento. Riprova.
30	BREAK	Il comando STOP è stato inserito in un programma o il tasto **STOP** è stato premuto per fermare la esecuzione del programma.
31	CAN'T RESUME	Incontrata frase senza istruzione TRAP effettiva.
32	LOOP NOT FOUND	Il programma ha incontrato un'istruzione DO e non trova il LOOP (ciclo) corrisp.
33	LOOP WITHOUT DO	Incontrato LOOP senza istruzione DO attiva.
34	DIRECT MODE ONLY	Questo comando è permesso solo in modo diretto, non da un programma.
35	NO GRAPHICS AREA	Un comando (DRAW, BOX, ecc) per creare la grafica è stato incontrato prima che fosse eseguito il comando GRAPHIC.
36	BAD DISK	Tentativo fallito di HEADER un disk perchè il metodo header (scheda di testata) veloce è stato tentato su dischetto non formattato o su dischetto difett.
37	BEND NOT FOUND	Il programma ha incontrato un "IF...THEN BEGIN" o "IF...THEN... ELSE BEGIN" e non ha trovato una parola chiave BEND che vada con BEGIN.
38	LINE NUMBER TOO LARGE	Errore nel rinumerare un programma BASIC. I parametri dati risultano in un numero di linea maggiore di 63999 generato; perciò, non è stata eseguita la rinumerazione.
39	UNRESOLVED REFERENCE	Errore nella rinumerazione di un programma BASIC. Non esiste il num di linea a cui ci si è riferiti come comando (cioè GOTO 999). Perciò, non è stata eseguita la rinumerazione.
40	UNIMPLEMENTED COMMAND	Incontrato un comando non supportato dal BASIC 7.0.
41	FILE READ	Incontrata una condizione di errore al caricamento o lettura di un programma o file dal drive (cioè apertura della porta del drive mentre si stava caricando un programma).

APPENDICE B

MESSAGGI DI ERRORI DOS

I seguenti messaggi di errori DOS sono ritornati attraverso le variabili DS e DS$. La variabile DS contiene solo il numero dell'errore e la variabile DS$ contiene il numero dell'errore, il messaggio dell'errore e qualsiasi numero della traccia o del settore corrispondente. NOTA: I numeri di messaggi di errori minori di 20 dovrebbero essere ignorati con l'eccezione di 01, che dà informazioni sul numero di file cancellati con il comando SCRATCH.

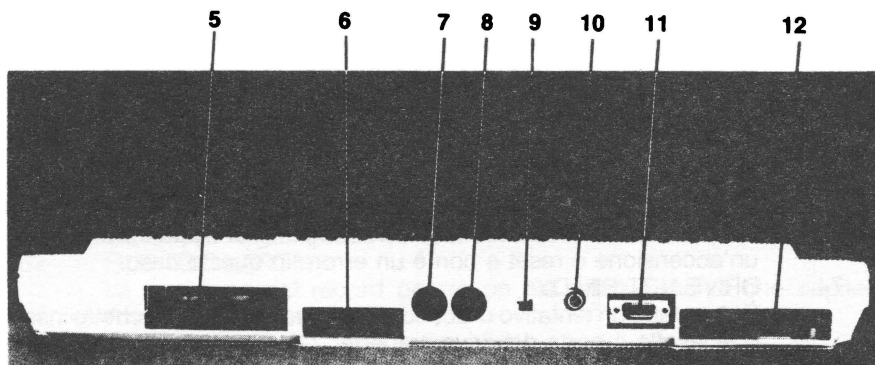
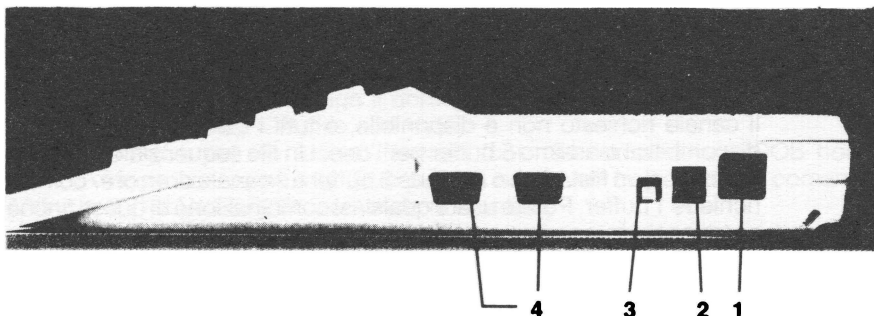
NUMERO DI ERRORE	MESSAGGIO E DESCRIZIONE
20	READ ERROR (blocco di testa non trovato) Il controllore del dischetto non è in grado di localizzare la testata del blocco del dato richiesto. Cause: un numero di settore non valido, o la testata è stata distrutta.
21	READ ERROR (no carattere di sincronismo) Il controllore del disco non è in grado di riconoscere un segno di sincronismo nella traccia desiderata. Cause: disallineamento della testina di lettura/scrittura, no disk, o dischetto non formattato o settato impropriamente. Può indicare anche un errore di hardware.
22	READ ERROR (blocco dati non presente) Si è richiesto al controllore del dischetto di leggere o verificare un blocco dei dati che non era scritto correttamente. Questo errore capita insieme ai comandi BLOCCO e può indicare una traccia non valida e/o una richiesta di settore.
23	READ ERROR (errore di controllo per somma nel blocco dei dati) Questo messaggio di errore indica che c'è un errore in uno o più byte dati. Il dato è stato letto nella memoria del DOS, ma il controllo per somma del dato è in errore. Questo messaggio può anche indicare problemi hardware di collegamento a massa.
24	READ ERROR (errore di decodificazione del byte) Il dato o la testata è stato letto nella memoria DOS ma è stato creato un errore hardware per una forma errata del bit nel byte del dato. Questo messaggio può anche indicare problemi hardware di collegamento a terra.
25	WRITE ERROR (errore di verifica scrittura) Questo messaggio è generato se il controllore riconosce una falsa connessione tra il dato scritto ed il dato nella memoria DOS.
26	WRITE PROTECT ON Questo messaggio è generato quando si richiede al controllore di scrivere un blocco di dati mentre il deviatore di protezione di scrittura è premuto. Ciò è causato dall'uso di un dischetto con un tabulato di protezione di scrittura di tacca o di un dischetto senza tacca.

NUMERO DI ERRORE	MESSAGGIO E DESCRIZIONE
27	READ ERROR Questo messaggio è generato quando è stato riconosciuto un errore di controllo per somma nella testata del blocco di dati richiesto. Il blocco non è stato letto nella memoria DOS.
28	WRITE ERROR Questo messaggio di errore è generato quando un blocco di dati è troppo lungo e ricopre il segno di sincron della testata successiva.
29	DISK ID MISMATCH Questo messaggio è generato quando si richiede al controllore di accedere ad un dischetto che non è stato inizializzato o che è stato formattato erratamente. Il messaggio può anche capitare se un dischetto ha una testata errata.
30	SYNTAX ERROR (sintassi generale) Il DOS non può interpretare il comando inviato al canale di comando. Questo è causato in modo tipico da un numero non valido di nomi di file, o da forme usate in modo non valido. Per esempio, due nomi di file appaiono nel lato sinistro del comando COPY.
31	SYNTAX ERROR (comando non valido) Il DOS non riconosce il comando. Il comando deve iniziare in prima posizione.
32	SYNTAX ERROR (comando non valido) Il comando inviato è più lungo di 58 caratteri Usate comandi del dischetto abbreviati.
33	SYNTAX ERROR (nome del file non valido) La corrispondenza della forma è usata in modo errato nei comandi OPEN o SAVE. Spiega il nome del file.
34	SYNTAX ERROR (file non dato) Il nome del file è stato lasciato fuori dal comando o il DOS non lo riconosce come tale. Due punti(:) sono stati tralasciati dal comando.
39	SYNTAX ERROR (comando non valido) Questo errore può risultare se il comando inviato al canale del comando (indirizzo second 15) non è riconosciuto dal DOS.
50	RECORD NOT PRESENT Risultato della lettura del dischetto dopo l'ultimo record attraverso i comandi INPUT# o GET#. Questo messaggio apparirà anche dopo il posizionamento in un record oltre la fine del file in un file relativo. Se l'intento è di espandere il file aggiungendo il nuovo record (con un comando PRINT#), il messaggio di errore può essere ignorato. INPUT# e GET# non dovranno essere tentati dopo il riconoscimento di questo errore senza prima il riposizionamento.
51	OVERFLOW IN RECORD L'istruzione PRINT# oltrepassa i limiti del record. L'informazione viene troncata. Poichè il ritorno carrello che è inviato come conclusione del record è contato nella grandezza del record, questo messaggio apparirà se il totale dei caratteri nel record (compreso il ritorno carrello finale) oltrepassa la grandezza definita del record.
52	FILE TOO LARGE La posizione del record dentro un file relativo indica che capiterà l'overflow del dischetto.
60	WRITE FILE OPEN Questo messaggio è generato quando un file di scrittura che non è stato chiuso viene aperto per la lettura.

NUMERO DI ERRORE	MESSAGGIO E DESCRIZIONE
61	FILE NOT OPEN Si accede ad un file che non è stato aperto nel DOS. Qualche volta in questa situazione, non viene generato il messaggio; la richiesta viene semplicemente ignorata.
62	FILE NOT FOUND Il file richiesto non esiste nel drive indicato.
63	FILE EXISTS Il nome del file che si sta creando esiste già nel dischetto.
64	FILE TYPE MISMATCH L'accesso al file richiesto non è possibile usando file del tipo nominato. Rileggete il capitolo che tratta quel tipo di file.
65	NO BLOCK Capita con l'allocazione del blocco. Il settore che avete tentato di allocare è già allocato. I numeri di traccia e di settore ritornati sono traccia e settore più alti successivi disponibili. Se il numero di traccia ritornato è 0, tutti i settori più alti che rimangono sono pieni. Se il dischetto non è ancora pieno, tentate traccia e settore più bassi.
66	ILLEGAL TRACK AND SECTOR Il DOS ha tentato di accedere ad una traccia o blocco che non esiste nel formato usato. Questo può indicare un problema nella lettura del puntatore del blocco successivo.
67	ILLEGAL SYSTEM T OR S Questo speciale messaggio di errore indica una traccia o settore del sistema non validi.
70	NO CHANNEL (disponibile) Il canale richiesto non è disponibile, o tutti i canali sono usati. Sono disponibili al massimo 5 buffer per l'uso. Un file sequenziale ha bisogno di 2 buffer; un file relativo richiede 3 buffer e il canale di errore/comando richiede 1 buffer. Potete usare qualsiasi combinazione di questi finchè la combinazione non supera i 5 buffer.
71	DIRECTORY ERROR La BAM (Mappa di Disponibilità del Blocco) del dischetto non corrisponde alla copia della memoria del dischetto. Per correggere, inizializzate il disk drive.
72	DISK FULL Sono stati usati i blocchi del dischetto o il directory è al limite di entrata. DISK FULL è inviato quando sono ancora disponibili due blocchi nel dischetto, per permettere al file corrente di essere chiuso.
73	DOS VERSION NUMBER (73,CBM DOS V30 1571,00,00) I DOS 1 e 2 sono compatibili per la lettura ma non per la scrittura. I dischetti possono essere letti in modo scambievole con entrambi i DOS, ma un disco formattato in una versione non può essere scritto con l'altra versione perchè il formato è diverso. Questo errore viene visualizzato tutte le volte che viene fatto un tentativo di scrivere su un dischetto che è stato formattato in un formato non compatibile. Apparirà anche dopo un'accensione o reset e non è un errore in questo caso.
74	DRIVE NOT READY È stato fatto un tentativo di accedere al drive senza un dischetto inserito; o la levetta o porta del drive è aperta.

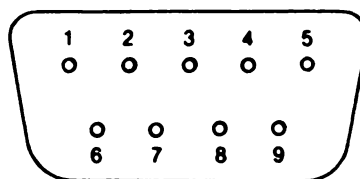
APPENDICE C

CONNETTORI/PORTE PER LA DOTAZIONE DELLE UNITÀ PERIFERICHE

1. Connettore Alimentazione
2. Commutatore Alimentazione
3. Tasto di Reset
4. Porte del Controllore
5. Porta di Espansione
6. Porta della Cassetta
7. Porta Seriale
8. Connettore del Video Composto
9. Selezionatore del Canale
10. Connettore RF
11. Connettore RGBI
12. Ingresso del l'Utente

COLLEGAMENTI LATERALI DEL PANNELLO

1. Connettore Alimentazione- La parte libera del cavo dell'alimentazione è attaccata qui.
2. Commutatore Alimentazione- Accende l'alimentazione dal trasformatore.
3. Tasto di Reset- Resetta il computer (partenza calda).
4. Porte del Controllore- Ci sono due porte del Controllore, numerate 1 e 2. Ogni porta del Controllore può accettare un joystick o mouse o un paddle di controllo dei giochi. Una penna ottica può essere inserita nella porta 1, la porta più vicina alla parte anteriore del computer. Usate le porte come da istruzioni del software.

PORTA 1 DEL CONTROLLORE

PIN	TIPO	NOTA
1	JOYA0	
2	JOYA1	
3	JOYA2	
4	JOYA3	
5	POT AY	
6	TASTO A/LP	
7	+5V	MAX.50mA
8	GND	
9	POT AX	

PORTA 2 DEL CONTROLLORE

PIN	TIPO	NOTA
1	JOYB0	
2	JOYB1	
3	JOYB2	
4	JOYB3	
5	POT BY	
6	TASTO B	
7	+5V	MAX.50mA
8	GND	
9	POT BX	

COLLEGAMENTI POSTERIORI

5. Porta di Espansione- Questo slot rettangolare è una porta parallela che accetta le cartucce di programma o di gioco e le interfacce speciali.

CARTUCCIA DELLA PORTA DI ESPANSIONE

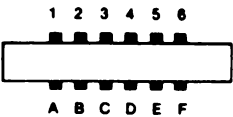
PIN	TIPO	PIN	TIPO
12	BA	1	GND
13	$\overline{\text{DMA}}$	2	+5V
14	D7	3	+5V
15	D6	4	$\overline{\text{IRQ}}$
16	D5	5	R/$\overline{\text{W}}$
17	D4	6	Clock Dot
18	D3	7	I/O 1
19	D2	8	$\overline{\text{GAME}}$
20	D1	9	$\overline{\text{EXROM}}$
21	D0	10	I/O 2
22	GND	11	$\overline{\text{ROML}}$

PIN	TIPO	PIN	TIPO
N	A9	A	GND
P	A8	B	$\overline{\text{ROMH}}$
R	A7	C	$\overline{\text{RESET}}$
S	A6	D	$\overline{\text{NMI}}$
T	A5	E	S 02
U	A4	F	A15
V	A3	H	A14
W	A2	J	A13
X	A1	K	A12
Y	A0	L	A11
Z	GND	M	A10

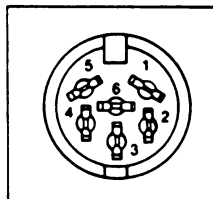
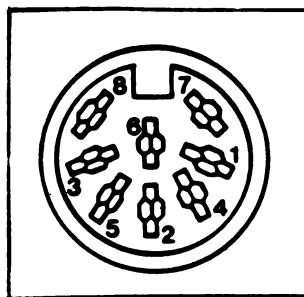
(visione della porta dal retro del C128)

6. Porta della Cassetta- Un registratore Datassette 1530 può essere collegato qui per memorizzare i programmi e le informazioni.

PORTA DELLA CASSETTA

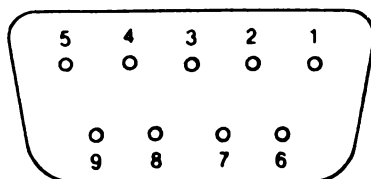
PIN	TIPO
A-1	GND
B-2	+5V
C-3	MOTORE DELLA CASSETTA
D-4	LETTURA DELLA CASSETTA
E-5	SCRITTURA DELLA CASSETTA
F-6	SENSORE DELLA CASSETTA

7. Porta Seriale- Una stampante seriale Commodore o un disk drive possono essere collegati direttamente al Commodore 128 attraverso questa porta.

PORTA DI I/O SERIALE

PIN	TIPO
1	SRQIN SERIALE
2	GND
3	ATN SERIALE IN/OUT
4	CLK SERIALE IN/OUT
5	DATO SERIALE IN/OUT
6	RESET

8. Connettore del Video Composto-Questo connettore DIN fornisce i segnali diretti di audio e video composto. Questi possono essere collegati al monitor Commodore o usati con componenti separati. Questo è il connettore di output a 40 colonne.

(visione della porta di fronte al retro del C128)

CONNETTORE DEL VIDEO COMPOSTO

PIN	TIPO	NOTA
1	LUM/SINC	Output di Luminosità/SINC
2	GND	
3	AUDIO OUT	
4	VIDEO OUT	Output del segnale composto
5	AUDIO IN	
6	COLORE OUT	Output del segnale di cromaticità
7	NC	No collegamento
8	NC	No collegamento

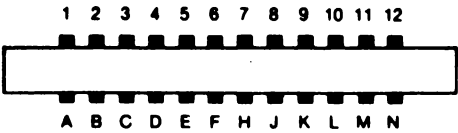
9. Selezionatore del Canale- Usate questo interruttore per selezionare su quale canale della TV (L-canale 3, H-canale 4) la figura del computer sarà visualizzata se usate un televisore invece di un monitor.
10. Connettore RF- Questo connettore fornisce sia le figure che il suono al vostro apparecchio televisivo. (Un televisore può visualizzare solo una figura di 40 colonne).
11. Connettore RGBI- Questo connettore di 9 pin fornisce un segnale di audio diretto e un segnale RGBI (Rosso/Verde (Green)/Blu/Intensità). Questo è l'output ad 80 colonne.

CONNETTORE RGBI

PIN	SEGNALE
1	Terra
2	Terra
3	Rosso
4	Verde
5	Blu
6	Intensità
7	Monocromatico
8	Sincronismo Orizzon.
9	Sincronismo Vert.

12. Ingresso dell'Utente- Qui possono essere collegate varie unità di interfaccia, incluso un modem Commodore.

PORTA DI I/O DELL'UTENTE

PIN	TIPO	NOTA
1	GND	
2	+5V	MAX.100mA
3	$\overline{\text{RESET}}$	
4	CNT1	
5	SP1	
6	CNT2	
7	SP2	
8	$\overline{\text{PC2}}$	
9	SER.ATN IN	
10	9 VAC	MAX.100mA
11	9 VAC	MAX.100mA
12	GND	

PIN	TIPO	NOTA
A	GND	
B	$\overline{\text{FLAG2}}$	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	GND	

APPENDICE D

CODICI DI VISUALIZZAZIONE DELLO SCHERMO

SCHERMO A 40 COLONNE

La Tabella che segue lista tutti i caratteri costruiti nei set di caratteri dello schermo del Commodore. Mostra quali numeri devono subire il POKE nella memoria dello schermo del chip VIC (40 colonne) (locazioni da 1024 a 2023) per ottenere il carattere desiderato sullo schermo a 40 colonne.
Per porre la memoria del colore, dovete usare le locazioni da 55296 a 56295). Mostra anche quale carattere corrisponde ad un numero che ha subito il PEEK dallo schermo.

Sono disponibili due set di caratteri. Sono disponibili entrambi nello stesso momento in modo 80 colonne, ma è disponibile solo uno alla volta in modo 40 colonne. I set sono commutabili tenendo premuti i tasti **SHIFT** e **Commodore** contemporaneamente. L'intero schermo di caratteri cambia nel set di caratteri selezionato. Dal BASIC, PRINT CHR$(142) cambierà i modi maiuscole/grafica e PRINT CHR$(14) i modi maiuscole/minuscole.

Qualsiasi numero della tabella può essere visualizzato anche in inverso. Il codice del carattere inverso si ottiene aggiungendo 128 ai valori mostrati.

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	K	k	11	V	v	22
A	a	1	L	l	12	W	w	23
B	b	2	M	m	13	X	x	24
C	c	3	N	n	14	Y	y	25
D	d	4	O	o	15	Z	z	26
E	e	5	P	p	16	[		27
F	f	6	Q	q	17	£		28
G	g	7	R	r	18	]		29
H	h	8	S	s	19	↑		30
I	i	9	T	t	20	←		31
J	j	10	U	u	21	SPACE		32

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
!		33		A	65			97
"		34		B	66			98
#		35		C	67			99
$		36		D	68			100
%		37		E	69			101
&		38		F	70			102
'		39		G	71			103
(		40		H	72			104
)		41		I	73			105
*		42		J	74			106
+		43		K	75			107
,		44		L	76			108
−		45		M	77			109
.		46		N	78			110
/		47		O	79			111
0		48		P	80			112
1		49		Q	81			113
2		50		R	82			114
3		51		S	83			115
4		52		T	84			116
5		53		U	85			117
6		54		V	86			118
7		55		W	87			119
8		56		X	88			120
9		57		Y	89			121
:		58		Z	90			122
;		59			91			123
<		60			92			124
=		61			93			125
>		62			94			126
?		63			95			127
		64	SPACE		96			

I codici dal 128 al 255 sono immagini inverse dei codici da 0 a 127.

APPENDICE E

CODICI ASCII E CHR$

Questa appendice vi mostra quali caratteri appariranno se digitate PRINT CHR$(X), per tutti i valori possibili di X. Mostra anche i valori ottenuti digitando PRINT ASC("x"), dove x è qualsiasi carattere che può essere visualizzato. Questo è utile per la valutazione del carattere ricevuto in una istruzione GET, nella conversione di maiuscole in minuscole e nella stampa di comandi con caratteri base (come la conversione in maius./min.) che non potrebbero essere incluse nelle istruzioni.

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0		23	.	46	E	69
	1		24	/	47	F	70
	2		25	0	48	G	71
	3		26	1	49	H	72
	4		27	2	50	I	73
WHT	5	RED	28	3	51	J	74
	6	CRSR →	29	4	52	K	75
	7	GRN	30	5	53	L	76
DISABLES SHIFT C=	8	RED	31	6	54	M	77
ENABLES SHIFT C=	9	SPACE	32	7	55	N	78
	10	!	33	8	56	O	79
	11	"	34	9	57	P	80
	12	#	35	:	58	Q	81
RETURN	13	$	36	;	59	R	82
SWITCH TO LOWER CASE	14	%	37	<	60	S	83
	15	&	38	=	61	T	84
	16	'	39	>	62	U	85
CRSR	17	(	40	?	63	V	86
RVS OFF	18	)	41	@	64	W	87
CLR HOME	19	•	42	A	65	X	88
INST DEL	20	+	43	B	66	Y	89
	21	,	44	C	67	Z	90
	22	−	45	D	68	[	91

PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS
£	92		115	f4	138		161
]	93		116	f6	139		162
↑	94		117	f8	140		163
←	95		118	SHIFT RETURN	141		164
	96		119	[illegible]	142		165
	97		120		143		166
	98		121	BLK	144		167
	99		122	↓ CRSR	145		168
	100		123	RVS OFF	146		169
	101		124	CLR HOME	147		170
	102		125	INST DEL	148		171
	103		126	Brown	149		172
	104		127	Lt. Red	150		173
	105		128	Dk. Gray	151		174
	106	Orange	129	Gray	152		175
	107		130	Lt. Green	153		176
	108		131	Lt. Blue	154		177
	109		132	Lt. Gray	155		178
	110	f1	133	RUN	156		179
	111	f3	134	← CRSR	157		180
	112	f5	135	YEL	158		181
	113	f7	136	CYN	159		182
	114	f2	137	SPACE	160		183

PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS
	184		186		188		190
	185		187		189		191

CODICI	**192-223**	**UGUALI A**	**96-127**
CODICI	**224-254**	**UGUALI A**	**160-190**
CODICI	**255**	**UGUALI A**	**126**

NOTA: L'output 80 colonne (RGBI) ha tre colori che sono diversi dall'output del colore a 40 colonne (video composto). Questo significa che i codici della stringa di caratteri che rappresentano i codici del colore per questi tre colori sono usati in modo diverso a seconda dell'output video usato. I seguenti codici di stringhe di caratteri rappresentano questi colori in ogni output video.

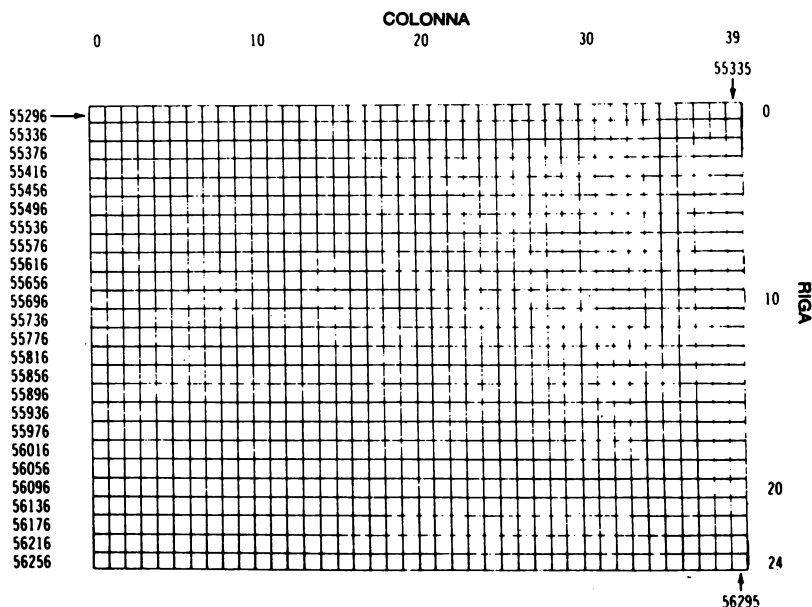
CHR$	40 COLONNE (VIC COMPOSTO)	80 COLONNE (RGBI 8563)
129	Arancione	Rosso Scuro
149	Marrone	Giallo Scuro
151	Grigio Scuro	Azzurro Scuro

APPENDICE F

MAPPE DELLA MEMORIA DELLO SCHERMO E DEL COLORE-MODO C128, 40 COLONNE E MODO C64

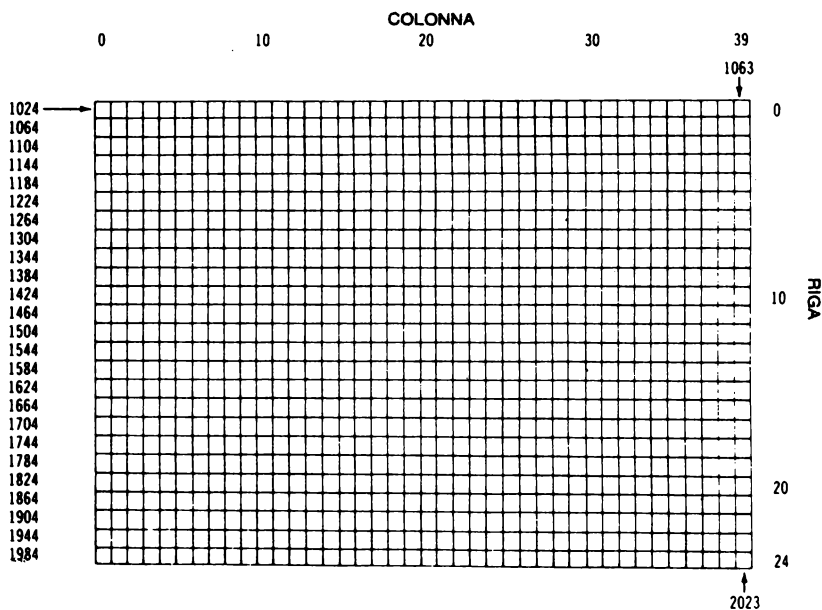
Le mappe che seguono visualizzano le locazioni della memoria usate in modo 40 colonne (C128 e C64) per l'identificazione dei caratteri dello schermo e del loro colore. Ogni mappa è controllata separatamente e consiste di 1000 posizioni.

Il carattere visualizzato sulle mappe può essere controllato direttamente con il comando POKE.

MAPPA DELLA MEMORIA DELLO SCHERMO DEL CHIP VIC (40 COLONNE)

La Mappa dello Schermo subisce il POKE con un valore del Codice di Visualizzazione dello Schermo (consultate l'Appendice D). Per esempio:

POKE 1024,13 (dal BANCO 0 o 15)

visualizzerà la lettera M nell'angolo in alto a sinistra dello schermo.

MAPPA DELLA MEMORIA DEL COLORE DEL CHIP VIC (40 COLONNE)

Se la mappa del colore subisce il POKE con un valore del colore, cambia il carattere colore. Per esempio:

POKE 552961,1 (dal BANCO 15)

cambierà la lettera M inserita prima dal verde chiaro al bianco.

CODICI DEL COLORE—40 COLONNE

0 Nero	**8 Arancione**
1 Bianco	**9 Marrone**
2 Rosso	**10 Rosso Chiaro**
3 Azzurro	**11 Grigio Scuro**
4 Porpora	**12 Grigio**
5 Verde	**13 Verde Chiaro**
6 Blu	**14 Blu Chiaro**
7 Giallo	**15 Grigio Chiaro**

Memoria di Controllo del Margine 53280
Memoria di Controllo dello sfondo 53281

APPENDICE G

FUNZIONI TRIGONOMETRICHE DERIVATE

FUNZIONE	EQUIVALENTE BASIC
secante	sec(x) = 1/cos(x)
cosecante	csc(x) = 1/sen(x)
cotangente	cot(x) = 1/tan(x)
inversa di seno	arcsen(x) = atn(x/sqr(x+1))
inversa di coseno	arccos(x) = -atn(x/sqr(-x*x+1))+π/2
inversa di secante	arcsec(x) = atn(x/sqr(x*x-1))
inversa di cosecante	arccsc(x) = atn(x/sqr(x*x-1))+(sgn(x)-1)* /2
inversa di cotangente	arccot(x) = -atn(x)+/2
seno iperbolico	senh(x) = (exp(x)-exp(-x))/2
coseno iperbolico	cosh(x) = (exp(x)+exp(-x))/2
tangente iperbolica	tanh(x) = -exp(-x)/(exp(x)+exp(-x))*2+1
secante iperbolica	sech(x) = 2/(exp(x)+exp(-x))
cosecante iperbolica	csch(x) = 2/(exp(x)-exp(-x))
cotangente iperbolica	coth(x) = exp(-x)/(exp(x)-exp(-x))*2+1
inversa di seno iperbolico	arcsenh(x) = log(x+ sqr(x*x+1))
inversa di coseno iperbolico	arccosh(x) = log(x+sqr(x*x-1))
inversa di tangente iperbolica	arctanh(x) = log(1+x)/ (1-x))/2
inversa di secante iperbolica	arcsech(x) = log(sqr (-x*x+1)+1/x)
inversa di cosecante iperbolica	arccsch(x) = log(sgn(x)*sqr(x*x+1)/x
inversa di cotangente iperbolica	arccoth(x) = log((x+1)/(x-1))/2

APPENDICE H

CODICI DI CONTROLLO ED ESCAPE

CODICI DI CONTROLLO

La tabella che segue lista i codici di controllo usati dal C 128. I codici di stampa nella prima colonna sono usati nelle istruzioni PRINT. I codici dei tasti nella seconda colonna sono la sequenza dei tasti digitati per eseguire controlli specifici. Tenete premuto il tasto **CONTROL** (o il tasto specificato a sinistra della colonna dei codici dei tasti) e digitate il tasto specificato a destra della colonna dei codici dei tasti.

STAMPA CODICI (CHR$)	CODICI DEI TASTI SEQUENZA TASTI	FUNZIONE	REALE IN MODO C64	C128
CHR$(2)	CTRL B	Sottolinea (80)	sì	sì
CHR$(5)	CTRL 2 o CTRL E	Pone colore car bianco	sì	sì
CHR$(7)	CTRL G	Produce il campanello	sì	sì
CHR$(8)	CTRL H	Disabil. cambio insieme car.	sì	sì
CHR$(9)	CTRL I	Abil. cambio insieme car.	sì	sì
		Muove cursore in pos. tab. successiva	sì	sì
CHR$(10)	CTRL J	Interlinea	sì	sì
CHR$(11)	CTRL K	Abil. cambio insieme car.	sì	sì
CHR$(12)	CTRL L	Disabil. cambio modo carattere	sì	sì
CHR$(13)	CTRL M	Pone ritorno carrello e interlinea al computer inserisce linea BASIC	sì	sì
CHR$(14)	CTRL N	Pone insieme car maiuscolo/minuscolo	sì	sì
CHR$(15)	CTRL O	Accende lampeggio (80)	sì	sì
CHR$(17)	CRSR DOWN CTRL Q	Muove il curs. giù una riga	sì	sì
CHR$(18)	CTRL 9 o CTRL R	Fa stampare car inversi	sì	sì
CHR$(19)	HOME	Muove curs. in pos. home (alto sin.) del video (finestra corrente)	sì	sì

NOTA: (40) = solo video a 40 colonne.
(80) = solo video a 80 colonne.

STAMPA CODICI (CHR$)	CODICI DEI TASTI SEQUENZA TASTI	FUNZIONE	REALE IN MODO C64	C128
CHR$(20)	DEL o CTRL T	Cancella ultimo tasto premuto e muove tutti car a destra del car cancellato uno spazio a sinistra	sì	sì
CHR$(24)	CTRL X, CTRL TAB o COMMODORE TAB	Tab poni/cancella	sì	sì
CHR$(27)	ESC o CTRL[	Invia car ESC	sì	sì
CHR$(28)	CTRL 3 o CTRL ?	Pone car colore rosso (40) e (80)	sì	sì
CHR$(29)	CRSR o CTRL]	Muove curs. di una colonna a destra	sì	sì
CHR$(30)	CTRL 6 o CTRL	Pone il colore del car verde (40) e (80)	sì	sì
CHR$(31)	CTRL 7 o CTRL =	Pone colore car blu (40) e (80)	sì	sì
CHR$(34)		Stampa frase doppia su schermo e pone editor in modo frase	sì	sì
CHR$(129)	COMMODORE 1	Pone colore carattere arancione (40); rosso scuro (80)	sì	sì
CHR$(130)		Sottolineat. off (80)	sì	sì
CHR$(131)		Gira program. questo codice CHR$ non funziona in PRINT CHR$ (131) ma funz. dal buffer della tastiera	sì	sì
CHR$(133)	F1	Codice CHR$ riservato per tasto F1	sì	sì
CHR$(134)	F3	Codice CHR$ riservato per tasto F3	sì	sì
CHR$(135)	F5	Codice CHR$ riservato per tasto F5	sì	sì
CHR$(136)	F7	Codice CHR$ riservato per tasto F7	sì	sì
CHR$(137)	F2	Codice CHR$ riservato per tasto F2	sì	sì
CHR$(138)	F4	Codice CHR$ riservato per tasto F4	sì	sì
CHR$(139)	F6	Codice CHR$ riservato per tasto F6	sì	sì
CHR$(140)	F8	Codice CHR$ riservato per tasto F8	sì	sì
CHR$(141)	SHIFT RETURN, CTRL ENTER, COMMODORE ENTER o RETURN	Invia un ritorno carrello e interlinea senza entrare in linea BASIC	sì	sì
CHR$(142)		Pone insieme car maiuscolo/grafico	sì	sì
CHR$(143)		Spegne lampeggio (80)	sì	sì

NOTA: (40) = schermo solo 40 colonne.
(80) = schermo solo 80 colonne.

STAMPA CODICI (CHR$)	CODICI DEI TASTI SEQUENZA TASTI	FUNZIONE	REALE IN MODO C64	C128
CHR$(144)	CTRL 1	Pone col car nero (40) e (80)	sì	sì
CHR$(145)	CRSR UP	Muove curs. in pos. stampa su di 1 riga	sì	sì
CHR$(146)	CTRL 0	Fine video inverso		
CHR$(147)	CLEAR HOME	Azzera finestra e muove curs. in pos. alto sin.	sì	sì
CHR$(148)	INST	Muove car da pos curs. a fine linea destra di 1 colonna	sì	sì
CHR$(149)	COMMODORE 2	Pone colore car marrone (40); giallo scuro (80)	sì	sì
CHR$(150)	COMMODORE 3	Pone colore car rosso chiaro (40) e (80)	sì	sì
CHR$(151)	COMMODORE 4	Pone colore car grigio scuro (40); azzurro scuro (80)	sì	sì
CHR$(152)	COMMODORE 5	Pone colore car grigio (40) e (80)	sì	sì
CHR$(153)	COMMODORE 6	Pone colore car verde chiaro (40) e (80)	sì	sì
CHR$(154)	COMMODORE 7	Pone colore car blu chiaro (40) e (80)	sì	sì
CHR$(155)	COMMODORE 8	Pone colore car grigio chiaro (40) e (80)	sì	sì
CHR$(156)	CTRL 5	Pone colore car porpora (40) e (80)	sì	sì
CHR$(157)	CRSR LEFT	Muove curs a sinistra di una colonna	sì	sì
CHR$(158)	CTRL 8	Pone colore car giallo (40) e (80)	sì	sì
CHR$(159)	o CTRL 4	Pone colore car azzurro (40); azzurro chiaro (80)	sì	sì

NOTA: (40) = solo schermo 40 colonne.
(80) = solo schermo 80 colonne..

CODICI ESCAPE

Questa tabella lista le sequenze di tasti per le funzioni ESCape disponibili sul Commodore 128. Le sequenze ESCape sono inserite premendo e lasciando il tasto **ESC**, poi premendo il tasto listato nella colonna di destra.

FUNZIONE ESCAPE	SEQUENZA TASTI
Cancella frase, inverso, lampeggio	ESC O
Cancella fino alla fine della linea corrente	ESC Q
Cancella inizio della linea corrente	ESC P
Cancella fino a fine schermo	ESC @
Muovi all'inizio della linea corrente	ESC J
Muovi alla fine della linea corrente	ESC K
Abilita modo auto inserimento	ESC A
Disabilita modo auto inserim.	ESC C
Cancella la linea corrente	ESC D
Inserisce una linea	ESC I
Pone tab stop per default (8 spazi)	ESC Y
Cancella tutti tab stop	ESC Z
Abilita lo scrolling	ESC L
Disabilita lo scrolling	ESC M
Scroll in su	ESC V
Scroll in giù	ESC W
Abilita campanello (control G)	ESC G
Disabilita campanello	ESC H
Pone cursore in modo non lampeggio	ESC E
Pone cursore in modo lampeggio	ESC F
Pone l'angolo in basso a destra della finestra nella posizione del cursore	ESC B
Pone l'angolo in alto a sinistra della finestra nella posizione del cursore	ESC T
Unità per lo spostamento 40/80 colonne dell'output dello schermo	ESC X

Le seguenti sequenze ESCape hanno effetto solo sullo schermo ad 80 colonne.

FUNZIONE ESCAPE	SEQUENZA TASTI
Cambia in cursore sottolineato (80)	ESC U
Cambia in cursore blocco (80)	ESC S
Pone lo schermo in video inverso (80)	ESC R
Ritorna lo schermo allo stato normale (non video inverso) (80)	ESC N

APPENDICE I

ABBREVIAZIONI BASIC 7.0

NOTA: Le abbreviazioni seguenti operano in modo maiuscole/grafica. Premete le lettere tasto(i) indicati, poi tenete premuto il tasto **SHIFT** e premete la lettera del tasto che segue la parola SHIFT.

PAROLA CHIAVE	ABBREVIAZIONE
ABS	A SHIFT B
APPEND	A SHIFT P
ASC	A SHIFT S
ATN	A SHIFT T
AUTO	A SHIFT U
BACKUP	BA SHIFT C
BANK	B SHIFT A
BEGIN	B SHIFT E
BEND	BE SHIFT N
BLOAD	B SHIFT L
BOOT	B SHIFT O
BOX	nessuna
BSAVE	B SHIFT S
BUMP	B SHIFT U
CATALOG	C SHIFT A
CHAR	CH SHIFT A
CHR$	C SHIFT H
CIRCLE	C SHIFT I
CLOSE	CL SHIFT O
CLR	C SHIFT L
CMD	C SHIFT M
COLLECT	COLL SHIFT E
COLLISION	COL SHIFT L
COLOR	COL SHIFT O
CONCAT	C SHIFT O
CONT	nessuna
COPY	CO SHIFT P
COS	nessuna
DATA	D SHIFT A
DEC	nessuna
DCLEAR	DCL SHIFT E
DCLOSE	D SHIFT C
DEF FN	nessuna

PAROLA CHIAVE	ABBREVIAZIONE
DELETE	DE SHIFT L
DIM	D SHIFT I
DIRECTORY	DI SHIFT R
DLOAD	D SHIFT L
DO	nessuna
DOPEN	D SHIFT O
DRAW	D SHIFT R
DS	nessuna
DS$	nessuna
DSAVE	D SHIFT S
DVERIFY	D SHIFT V
EL	nessuna
END	nessuna
ENVELOPE	E SHIFT N
ER	nessuna
ERR$	E SHIFT R
EXIT	EX SHIFT I
EXP	E SHIFT X
FAST	nessuna
FETCH	F SHIFT E
FILTER	F SHIFT I
FOR	F SHIFT O
FRE	F SHIFT R
FNXX	nessuna
GET	G SHIFT E
GETKEY	GETK SHIFT E
GET#	nessuna
GOSUB	GO SHIFT S
GO64	nessuna
GOTO	G SHIFT O
GRAPHIC	G SHIFT R
GSHAPE	G SHIFT S
HEADER	HE SHIFT A
HELP	X SHIFT X
HEX$	H SHIFT E
IF...GOTO	nessuna
IF...THEN...ELSE	nessuna
INPUT	nessuna
INPUT#	I SHIFT N
INSTR	IN SHIFT S
INT	nessuna
JOY	J SHIFT O
KEY	K SHIFT E
LEFT$	LE SHIFT F
LEN	nessuna
LET	L SHIFT E
LIST	L SHIFT I
LOAD	L SHIFT O
LOCATE	LO SHIFT C
LOG	nessuna
LOOP	LO SHIFT O

PAROLA CHIAVE	ABBREVIAZIONE
MID$	M SHIFT I
MONITOR	MO SHIFT N
MOVSPR	M SHIFT O
NEW	nessuna
NEXT	N SHIFT E
ON GOSUB	ON GO SHIFT S
ON GOTO	ON G SHIFT O
OPEN	O SHIFT P
PAINT	P SHIFT A
PEEK	PE SHIFT E
PEN	P SHIFT E
PI	nessuna
PLAY	P SHIFT L
POINTER	PO SHIFT I
POKE	PO SHIFT K
POS	nessuna
POT	P SHIFT O
PRINT	nessuna
PRINT#	P SHIFT R
PRINT USING	US SHIFT I
PUDEF	P SHIFT U
RCLR	R SHIFT C
RDOT	R SHIFT D
READ	RE SHIFT A
RECORD	R SHIFT E
REM	nessuna
RENAME	RE SHIFT N
RENUMBER	REN SHIFT U
RESTORE	RE SHIFT S
RESUME	RES SHIFT U
RETURN	RE SHIFT T
RGR	R SHIFT G
RIGHT$	R SHIFT I
RND	R SHIFT N
RREG	R SHIFT R
RSPCOLOR	RSP SHIFT C
RSPPOS	R SHIFT S
RSPR	nessuna
RSPRITE	RSP SHIFT R
RUN	R SHIFT U
RWINDOW	R SHIFT W
SAVE	S SHIFT A
SCALE	SC SHIFT A
SCNCLR	S SHIFT C
SCRATCH	SC SHIFT R
SGN	S SHIFT G
SIN	S SHIFT I
SLEEP	S SHIFT L
SLOW	nessuna
SOUND	S SHIFT O
SPC	nessuna

PAROLA CHIAVE	ABBREVIAZIONE
SPRCOLOR	SPR SHIFT C
SPRDEF	SPR SHIFT D
SPRITE	S SHIFT P
SPRSAV	SPR SHIFT S
SQR	S SHIFT Q
SSHAPE	S SHIFT S
STASH	S SHIFT T
ST	nessuna
STEP	ST SHIFT E
STOP	ST SHIFT O
STR$	ST SHIFT R
SWAP	S SHIFT W
SYS	nessuna
TAB(	T SHIFT A
TAN	nessuna
TEMPO	T SHIFT E
TI	nessuna
TI$	nessuna
TO	nessuna
TRAP	T SHIFT R
TROFF	TRO SHIFT F
TRON	TR SHIFT O
UNTIL	U SHIFT N
USR	U SHIFT S
VAL	nessuna
VERIFY	V SHIFT E
VOL	V SHIFT O
WAIT	W SHIFT A
WHILE	W SHIFT H
WIDTH	WI SHIFT D
WINDOW	W SHIFT I
XOR	X SHIFT O

APPENDICE J

RIASSUNTO DEI COMANDI DEL DISCHETTO

Questa appendice lista i comandi usati per il funzionamento del dischetto nei modi C128 e C64 sul Commodore 128. Per informazioni dettagliate su uno qualsiasi di questi comandi, consultate il Capitolo 3. Anche il manuale del vostro disk drive contiene informazioni sui comandi del dischetto.

I nuovi comandi **BASIC 7.0** possono essere usati solo in modo C128. Tutti i comandi BASIC 2.0 possono essere usati sia in modo C128 che in modo C64.

COMANDO	USO	BASIC 2.0	BASIC 7.0
APPEND	Appone il dato al file		sì
BLOAD	Carica un file binario che inizia nella locaz di memoria specificata		sì
BOOT	Carica ed esegue un programma che può subire il boot		sì
BSAVE	Salva un file binario dalla locaz di memoria specificata		sì
CATALOG	Visualizza il conten del directory del dischetto su schermo*		sì
CLOSE	Chiude il file logico del dischetto	sì	
CMD	Ridirige l'output dello schermo ad una unità periferica	sì	
COLLECT	Libera lo spazio inaccessibile del dischetto*		sì
CONCAT	Concatena due file di dati*		sì
COPY	Copia i file tra le unità*		sì
DCLEAR	Azzera tutti i canali aperti del drive		sì
DCLOSE	Chiude i file logici del dischetto		sì
DIRECTORY	Visualizza il directory dei contenuti del dischetto su schermo*		sì
DLOAD	Carica un programma in BASIC dal dischetto		sì
DOPEN	Apre un file per una operazione di lettura e/o scrittura		sì
DSAVE	Salva un programma in BASIC nel dischetto		sì
DVERIFY	Verifica il programma in memoria con il programma su dischetto		sì
GET#	Riceve l'input da un file aperto del disco	sì	

COMANDO	USO	BASIC 2.0	BASIC 7.0
HEADER	Formatta un dischetto*		sì
LOAD	Carica un file da dischetto	sì	
OPEN	Apre un file per input o output	sì	
PRINT#	Mette in output un dato verso il file	sì	
RECORD	Posiziona i puntatori dei file relativi*		sì
RENAME	Cambia il nome di un file su dischetto*		sì
RUN	Esegue un programma filename BASIC dal dischetto		sì
SAVE	Memorizza il programma in memoria nel dischetto	sì	
VERIFY	Verifica programma in memoria con il programma su disco	sì	

*Sebbene non ci sia un comando singolo equivalente in BASIC 2.0, c'è un'istruzione equivalente multi comando. Consultate il manuale del vostro disk drive per queste convenzioni BASIC 2.0.

APPENDICE K - PARTE I

CP/M DEL COMMODORE 128

Questa appendice spiega ciascuna routine CP/M BIOS, BIOS 8502 e delle Funzioni dell'Utente e come chiamare ciascuna di esse in linguaggio assembly (di assemblaggio) Z80. Questa sezione dà per scontato che abbiate già conoscenze del linguaggio macchina Z80 e delle operazioni di base del sistema CP/M.

La Parte I di questa appendice lista come prima cosa ciascuna routine CP/M BIOS, BIOS 8502 e delle Funzioni dell'Utente per numero. La Parte II spiega come chiamare queste routine e dà esempi. La Parte III lista la mappa della memoria dello Z80.

Il formato usato per descrivere queste routine è il seguente:

a) Nome della Funzione
b) Parametri di Input
c) Parametri di Output
d) Breve descrizione
e) Altre routine di preparazione richieste/post routine (o ulteriori informazioni)

Le routine BIOS 8502 e delle Funzioni dell'Utente richiedono che certi valori siano posti nei registri del microprocessore Z80. Nel Volume 2, Capitolo 6, avete imparato i registri del microprocessore 8502: A, X, Y, Stato (PSW), Puntatore dello Stack (S) ed il Contatore di Programma (PC). Anche lo Z80 ha registri applicabili. I registri dello Z80 sono chiamati nel seguente modo:

A	(Accumulatore)
BC	
DE	
HL	
PSW	(Parola di Stato)
IX	(Registro X)
IY	(Registro di indice Y)
PC	(Contatore di programma)
SP	(Puntatore dello stack)

Alcuni registri possono essere usati come coppia di registri, per rappresentare un indirizzo a 16 bit o come registro singolo di 8 bit. Lo Z80 ha un insieme duplicato di registri per il processo di interruzione.

ROUTINE CP/M BIOS DEL COMMODORE 128

Il sistema CP/M del Commodore 128 ha un insieme di routine chiamate CP/M BIOS, che gestiscono le operazioni a basso livello di input/output del sistema. Ognuna di queste routine può essere raggiunta per mezzo della tabella di salto CP/M BIOS sottostante. I numeri dei vettori di salto da 0 a 29 sono i vettori di salto CP/M BIOS. Il 30esimo vettore di salto è per le Funzioni dell'Utente dipendenti dal sistema. Esse sono trattate seguendo le routine CP/M BIOS.

N.	ISTRUZIONE	DESCRIZIONE
0	JMP BOOT	Esegue l'inizializzaz. a partenza fredda
1	JMP WBOOT	Esegue l'inizializzaz. a partenza calda
2	JMP CONST	Controlla carattere pronto per l'input da console
3	JMP CONIN	Legge Carattere della Console in
4	JMP CONOUT	Scrive Carattere Console out
5	JMP LIST	Scrive Carattere di Lista out
6	JMP AUXOUT	Scrive Carattere Ausiliario di Output
7	JMP AUXIN	Legge Carattere Ausiliario di Input
8	JMP HOME	Muove alla Traccia 00 del Dischetto Selez.
9	JMP SELDSK	Seleziona il Drive
10	JMP SETTRK	Pone il Numero della Traccia
11	JMP SETSEC	Pone il Numero del Settore
12	JMP SETDMA	Pone Indirizzo DMA
13	JMP READ	Legge il Settore Specificato
14	JMP WRITE	Scrive il Settore Specificato
15	JMP LISTST	Ritorna lo Stato di Lista
16	JMP SECTRN	Traduce il Settore Logico in quello Fisico
17	JMP CONOST	Ritorna lo Stato dell'Output della Console
18	JMP AUXIST	Ritorna lo Stato di Input della Porta Ausiliaria
19	JMP AUXOST	Ritorna lo Stato di Output della Porta Ausiliaria
20	JMP DEVTBL	Ritorna l'Indirizzo della Tabella di I/O dei Caratteri
21	JMP DEVINI	Inizializza le Unità di I/O Caratteri
22	JMP DRVTBL	Ritorna l'Indirizzo della Tabella del Disk Drive
23	JMP MULTIO	Pone il Numero dei settori Logicamente Consecutivi da leggere o scrivere
24	JMP FLUSH	Forza lo Scorrimento del Buffer Fisico per lo Sblocco supportato dall'Utente
25	JMP MOVE	Movimento Memoria verso Memoria
26	JMP TIME	Segnale di Poni/Ottieni Tempo
27	JMP SELMEN	Seleziona il Banco di Memoria
28	JMP SETBNK	Specifica il Banco per un'Operazione DMA
29	JMP XMOVE	Pone il Banco se un Buffer è in un Banco diverso da 0 o 1
30	JMP USERF	Riservato per l'Implementatore del Sistema
31	JMP RESERV1	Riservato per un Uso Futuro
32	JMP RESERV2	Riservato per un Uso Futuro

Tabella dei Vettori di Salto CP/M 3 BIOS

Le routine BIOS 8502 sono un sottoinsieme delle Funzioni dell'Utente dipendenti dal sistema (specificatamente, la Funzione dell'Utente numero 4) contenuta nel sistema CP/M del Commodore 128. Molte Funzioni dell'Utente hanno sottofunzioni che richiedono che certi parametri passino attraverso i registri dello Z80. Questo viene trattato in dettaglio nella sezione delle Funzioni dell'Utente in questa appendice. Esempi di richiamo delle routine CP/M BIOS e delle Funzioni dell'Utente sono fornite nella Parte II di questa appendice.

0 BOOT

Banco:	0
Input:	Nessuno
Output	Nessuno
Funzione:	Questo codice esegue tutta l'inizializzazione hardware, pone la pagina a zero, stampa qualsiasi messaggio segnato e carica il CCP poi trasferisce il controllo ai CCP.

1 WBOOT

Banco:	0 o 1
Input:	Nessuno
Output:	Nessuno
Funzione:	Questo codice predispone la pagina zero, ricarica il CCP e poi esegue il CCP.

2 CONST

Banco:	0 o 1
Input:	Nessuno
Output:	A = 0FFH se carattere della console A = 00H se non carattere della console
Funzione:	Controlla lo stato di input di console delle unità correnti della console. Se un qualsiasi delle unità ha un carattere disponibile, FFH viene ritornato, altrimenti viene ritornato 00H.

3 CONIN

Banco:	0 o 1
Input:	Nessuno
Output:	A = carattere ASCII di console
Funzione:	Legge un carattere da UNA qualsiasi unità di input di console assegnata. Una scansione di ciascuna unità assegnata viene fatta finchè si trova un carattere di input. Il carattere è ritornato nel registro A.

4 CONOUT

Banco:	0 o 1
Input:	C = carattere ASCII da visualizzare
Output:	Nessuno
Funzione:	Invia il carattere in C a TUTTE le unità che sono assegnate correntemente alla console. Aspetta che tutte le unità assegnate accettino un carattere prima di uscire.

5 LIST
- Banco: 0 o 1
- Input: C = carattere ASCII da stampare
- Output: Nessuno
- Funzione: Invia il carattere in C a TUTTE le unità che sono correntemente assegnate alla unità LIST. Aspetta che tutte le unità assegnate accettino un carattere prima di uscire.

6 AUXOUT
- Banco: 0 o 1
- Input: C = carattere ASCII da inviare all'unità AUX
- Output: Nessuno
- Funzione: Invia il carattere C a TUTTE le unità assegnate correntem. all'unità AUXOUT. Aspetta che tutte le unità assegnate accettino un carattere prima di uscire.

7 AUXIN
- Banco: 0 o 1
- Input: Nessuno
- Output: A = carattere ASCII dall'unità AUX
- Funzione: Legge un carattere da UNA qualsiasi delle unità assegnate AUXIN. Una scansione di ogni unità assegnata viene fatta finchè viene trovato un carattere di input. Il carattere è ritornato nel registro A.

8 HOME
- Banco: 0
- Input: Nessuno
- Output: Nessuno
- Funzione: Porta in posizione home la testina del drive selezionato correntemente. Questa funzione pone la traccia corrente a 0 e non muove la testina del dischetto.

9 SELDSK
- Banco: 0
- Input: C = Disk Drive (0-15) (A = 0)
E = Flag Iniziale di Selezione (LSB)
- Output: HL = Indirizzo della Testata del Parametro del Dischetto (DPH) se esiste il drive, HL = 000H se non esiste.
- Funzione: Seleziona il drive il cui indirizzo è in C come drive corrente per tutte le ulteriori operazioni del dischetto. Se LSB del registro E è zero, allora questo è la prima registrazione di questo dischetto. Il tipo di disco (C64 CP/M, MFM o C128 CP/M) viene controllato ed i parametri DPB aggiustati per il dischetto che attualmente è nel drive.

10 SETTRK
- Banco: 0
- Input: BC = numero della traccia
- Output: Nessuno
- Funzione: La coppia di registri BC contiene il numero di traccia che si deve usare nell'accesso successivo al dischetto. Questo valore è salvato.

11 SETSEC

Banco:	0
Input:	BC = numero del settore
Output:	Nessuno
Funzione:	La coppia di registri BC contiene il numero del settore che deve essere usato nell'accesso successivo al dischetto. Questo valore viene salvato. Quello in BC è il valore ritornato dalla routine di traduzione del settore (in HL).

12 SETDMA

Banco:	0
Input:	BC = indirizzo di accesso diretto alla memoria
Output:	Nessuno
Funzione:	Il valore in BC viene salvato come indirizzo corrente di DMA. Questo è l'indirizzo dove avvengono TUTTE le letture o scritture dei dischetti. L'indirizzo DMA che viene posto è usato finchè è modificato da una chiamata futura in questa routine che lo cambia.

13 READ

Banco:	0
Input:	Nessuno
Output:	A = 000H se non ci sono errori A = 001H se c'è un errore non recuperabile A = 0FFH se sono cambiati i mezzi
Funzione:	Legge il settore indirizzato dal dischetto, traccia e settore correnti nell'indirizzo corrente DMA. Se il dato viene letto senza errori allora al ritorno A = 0. Se capita un errore, l'operaz. viene tentata molte volte e se non avviene una lettura corretta allora A è posto a 001H. Un test per il cambiamento dei mezzi viene eseguito ogni volta che questa routine è chiamata ed A è posta a -1 se i mezzi sono stati cambiati.

14 WRITE

Banco:	0
Input:	C = codice di sblocco (non usato)
Output:	A = 000H se non ci sono errori A = 001H se c'è un errore non recuperabile A = 002H se il dischetto è di sola lettura A = 0FFH se i mezzi sono cambiati
Funzione:	Scrive il settore indirizzato da parte di traccia e settore del dischetto corrente dall'indirizzo corrente DMA. Se il dato è scritto senza errori, allora A è posto a 0 al ritorno. Se capita un errore, l'operazione viene tentata parecchie volte e se non c'è una scrittura corretta, allora A è posto a 001H. Viene fatto un esame del cambiamento dei mezzi ogni volta che questa routine viene chiamata ed A è posto a -1 se i mezzi sono cambiati. Anche se viene fatto un tentativo di scrivere su un dischetto a sola lettura, allora il registro A è posto a 002H.

15 LISTST

Banco:	0 o 1
Input:	Nessuno
Output:	A = 00H se l'unità della lista non è pronta ad accettare un carattere. A = 0FFH se l'unità della lista è pronta ad accettare un carattere.
Funzione:	Questa routine esegue la scansione delle unità della lista assegnate correntemente e ritorna A posto a 0FFH se TUTTE le unità assegnate sono pronte ad accettare un carattere. Se qualsiasi unità assegnata non è pronta allora A è posto a 00H.

16 SECTRN

Banco:	0
Input:	BC = numero del settore logico (0-n) DE = indirizzo della tabella di traduzione (da DPB)
Output:	HL = numero del settore fisico
Funzione:	Questa routine converte il numero del settore fisico in numero del settore logico. Se non è necessaria la traduzione allora muove il registro BC in HL e ritorna.

17 CONOST

Banco:	0 o 1
Input:	Nessuno
Output:	A = 0FFH se pronto A = 000H se non pronto
Funzione:	Questa routine fa la scansione delle unità di console assegnate correntem. e ritorna con A posto a 0FFH se TUTTE le unità assegnate sono pronte ad accettare un carattere. Se qualsiasi unità assegnata non è pronta allora A è posto a 000H.

18 AUXIST

Banco:	0 o 1
Input:	Nessuno
Output:	A = 0FFH se è presente un carattere di console A = 000H se non c'è carattere di console
Funzione:	Controlla lo stato della unità corrente AUXIN. Se una delle unità ha un carattere disponibile, 0FFH è ritornato, altrimenti è ritornato 000H.

19 AUXOST

Banco:	0 o 1
Input:	Nessuno
Output:	A = 0FFH se pronto A = 000H se non è pronto
Funzione:	Questa routine fa la scansione delle unità AUXOUT assegnate correntem. e ritorna con A posto a 0FFH se TUTTE le unità sono pronte ad accettare un carattere. Se una delle unità assegnate non è pronta allora A è posto a 000H.

20 DEVTBL

Banco:	0 o 1
Input:	Nessuno
Output:	HL = indirizzo della tabella di I/O dei caratteri
Funzione:	Questa routine ritorna l'indirizzo della tabella di I/O dei caratteri. Questa tabella è usata per nominare ciascun modulo del driver (circuito pilota) e per porre /controllare la baud rate e la logica XON/XOFF per ogni driver. Nota: il meccanismo dell'unità drive è usato per sostituire IOBYTE usato con il CP/M 2.2.

21 DEVINI

Banco:	0 o 1
Input:	C = numero dell'unità
Output:	Nessuno
Funzione:	Inizializza l'unità a carattere fisico specificata nel registro C nella BAUD rate di DEVTBL.

22 DRVTBL

Banco:	0
Input:	Nessuno
Output:	HL = indirizzo della tabella del drive
Funzione:	Ritorna l'indirizzo della tabella del drive in HL (NOTA: la prima istruzione DEVE essere LXI H, DRVTBL). La tabella del drive è una lista di 16 puntatori delle parole che puntano al DPH per quel drive. Se un drive non è presente nel sistema, allora il puntatore di quel drive è posto a zero.

23 MULTIO

Banco:	0
Input:	C = conto del settore multiplo
Output:	Nessuno
Funzione:	Il conto del settore multiplo è posto prima dell'indirizzo della traccia, settore e DMA e prima che avvenga la lettura/scrittura dei settori. Può essere trasferito un massimo di 16K da ogni conto del settore multiplo.

24 FLUSH

Banco:	0
Input:	Nessuno
Output:	A = 000H se non ci sono errori A = 001H se ci sono errori non recuperabili A = 002H se il dischetto è di sola lettura A = 0FFH se sono cambiati i mezzi
Funzione:	Questa routine è usata solo se viene fatto il blocco/sblocco nel BIOS. Questo codice ritorna SEMPRE con A = 000H.

25 MOVE

Banco	0 o 1
Input:	HL = indirizzo di destinazione DE = indirizzo sorgente BC = conto
Output:	HL = HL(in) + BC(in) DE = DE(in) + BC(in)
Funzione:	Muove un blocco di dati. Il dato da muovere è in/da il banco corrente della memoria (o comune) a meno che non venga chiamata prima la routine XMOVE, allora c'è un movimento di dati tra i banchi.

26 TIME

Banco:	0 o 1
Input:	C = 000H (Ottieni Tempo)/ 0FFH (Poni Tempo)
Output:	Nessuno
Funzione:	Questa funzione è chiamata con C = 000H se il tempo del sistema in SCB necessita di essere aggiornato dal clock. Se C = 0FFH, allora il tempo in SCB è stato appena aggiornato ed il clock è posto al tempo SCB. NOTA: HL e DE DEVONO essere mantenuti.

27 SELMEM

Banco:	0 o 1
Input:	A = banco di memoria
Output:	Nessuno
Funzione:	Cambia il banco corrente di memoria. Questo codice DEVE essere nella memoria comune.

NOTA: SOLO A può essere modificato.

28 SETBNK

Banco:	0
Input:	A = banco di memoria DMA
Output:	Nessuno
Funzione:	Pone il banco DMA per l'operazione successiva di LETTURA/SCRITTURA.

29 XMOVE

Banco:	0
Input:	B = banco di destinazione C = banco sorgente
Output:	Nessuno
Funzione:	Dà al sistema la capacità di eseguire un DMA memoria verso memoria per l'intero spazio del sistema.

30 USERF

Banco:	0 o 1
Input:	A = numero della funzione, L = numero della sottofunzione
Output:	Gli output dipendono dalle funzioni o sottofunzioni chiamate.
Funzione:	Chiama le funzioni dell'utente e le routine BIOS 8502 che sono definite nella sezione delle Funzioni dell'Utente Dipendenti dal Sistema nel Commodore 128.

31 RESERV1

Banco:	N/A
Input:	N/A
Output:	N/A
Funzione:	Non disponibile per l'utente.

32 RESERV2

Banco:	N/A
Input:	N/A
Output:	N/A
Funzione:	Non disponibile per l'utente.

STRUTTURE DEI DATI

BLOCCO DI CONTROLLO DEL SISTEMA-(SCB)

Il Blocco di Controllo del Sistema è una struttura dei dati a 100 byte. La struttura dei dati è usata come comunicazione di base tra i vari moduli che costituiscono il sistema CP/M Plus. I contenuti della struttura dei dati sono parametri e variabili del sistema.

TABELLA DEL DRIVE (DRVTBL)

DA PH0 A DPH15

Una lista di 16 puntatori delle parole (formato byte inverso). Il primo puntatore (DPH0) è il drive A e l'ultimo puntatore (DPH15) è il drive P. I puntatori puntano a XDPH per quel drive. Qualsiasi drive che non è nel sistema ha i puntatori posti a zero.

PARAMETRI ESTESI DI TESTATA DEL DISCHETTO (XDPH) (DPH NORMALE CON UNA TESTATA)

WRT	READ	LOG-IN	INIT	TYPE	UNIT	XLT	-0-	MF	DPB	CSV	ALV	DIR-BCB	DTA BCB	HASH	H-BANK
16b	16b	16b	16b	8b	8b	16b	72b	8b	16b	16b	16b	16b	16b	16b	8b
-10	-8	-6	-4	-2	-1	0	2	11	12	14	16	18	20	22	24

WRT	Contiene l'indirizzo della routine di scrittura del settore per questo drive.
READ	Contiene l'indirizzo della routine di lettura del settore per questo drive.
LOGIN	Contiene l'indirizzo della routine di login (registrazione) per questo drive.
INIT	Contiene l'indirizzo della routine di prima inizializzaz. per questo drive.
TYPE	Questo byte è usato dal BIOS per controllare la densità e il tipo di mezzi.
UNIT	Contiene il numero del drive relativo al controllore del dischetto.
XLT	Contiene l'indirizzo della tabella di traduzione del settore o zero se non ce ne sono.
-0-	Area di lavoro BDOS (9 byte).
MF	Flag dei mezzi azzerato se il dischetto è registrato. BIOS è posto a 0FFH se i mezzi sono cambiati.
DPB	Contiene un puntatore del DPB corrente che descrive il tipo di mezzo corrente.
CSV	Contiene un puntatore dell'area di totale di controllo del directory (uno per drive).
ALV	Contiene un puntatore dell'area di allocazione dei vettori (uno per drive).
DIRBCB	Contiene un puntatore di un singolo Blocco di Controllo del Buffer del directory (BCB).
DTSBCB	Contiene un puntatore di un singolo Blocco di Controllo del Buffer di dati.
HASH	Contiene un puntatore di una tabella facoltativa del segno di quantità del directory (FFFFH non è usato).
HBANK	Contiene un numero di banco della tabella del segno di quantità del directory.

BLOCCO DEI PARAMETRI DEL DISCHETTO-DPB

SPT	BSH	BLM	EXM	DSM	DRM	AL0	AL1	CKS	OFF	PSH	PHM
16b	8b	8b	8b	16b	16b	8b	8b	16b	16b	8b	8b

SPT	Numero dei 128 record per traccia
BSH	Fattore di traslazione del blocco di allocazione dei dati
BLM	Maschera del blocco
EXM	Maschera di estensione
DSM	Numero del blocco di allocazione su dischetto meno uno
DRM	Numero delle entrate del directory meno una
AL0	Primo byte: Vettore di allocaz del blocco del directory. Riempito da MSB a LSB
AL1	Secondo byte: (possono essere usati fino a 16 blocchi di allocazione per il directory)
CKS	Misura del vettore di controllo del directory, (DRM+1)/4
OFF	Numero delle tracce riservate all'inizio del dischetto
PHS	Fattore di traslazione del record fisico
PHM	Maschera del record fisico

BLOCCO DI CONTROLLO DEL BUFFER (BLOCCO DI CONTROLLO LRU-BCB)

DRV	REC#	WFLG	00	TRACCIA	SETTORE	BUFFAD	BANCO	LINK
8b	24b	8b	8b	16b	16b	16b	8b	16b

DRV	Drive associato a questo record. Posto a 0FFH se non è usato.
REC#	Contiene il numero assoluto di settore del buffer.
WFLG	Posto a 0FFH quando il buffer contiene i dati che devono essere scritti nel dischetto.
00	Byte scratch (di lavoro) usato da BDOS.
TRACK	Indirizzo della traccia fisica del buffer.
SECTOR	Indirizzo del settore fisico del buffer.
BUFFAD	Indirizzo del buffer associato a questo BCB.
BANK	Numero di banco del buffer associato a questo BCB.
LINK	Contiene l'indirizzo del BCB successivo in questa lista unita. Posto a zero se ultimo.

FUNZIONI DELL'UTENTE DIPENDENTI DAL SISTEMA COMMODORE 128

Le routine in linguaggio macchina Z80 trattate in questa sezione si riferiscono esplicitamente al Personal Computer Commodore 128. Queste routine non sono parte del sistema CP/M standard che può passare da un computer di un produttore ad un altro. Essi sono scritti solo per essere lanciati sul C128.

La maggior parte di queste routine Z80 progettate per l'esigenza del C128 richiedono che certi parametri siano passati dall'utente ai registri appropriati dello Z80. Tutte queste routine, listate dal numero della funzione, richiedono che il numero della funzione sia posto nel registro A dello Z80 (accumulatore). Molte di queste routine hanno sottofunzioni che richiedono che l'utente ponga un valore nel registro L ad 8 bit, che viene usato per chiamare la sottofunzione appropriata. Tutti gli altri parametri richiesti in più sono rilevati se necessari. Questa sezione segue lo stesso formato della sezione precedente sul CP/M BIOS:

a) Nome della Funzione
b) Parametri di Input
c) Parametri di Output
d) Breve Descrizione
e) Ulteriori Informazioni

0 a) Funzione 0 dell'Utente: Legge qualsiasi Byte nel Banco 00 della RAM
  b) A=0 (numero della funzione)
  DE=indirizzo a 16 bit da leggere
  c) C=valore letto dall'indirizzo nel Banco 0 della RAM
  A=0 al RITorno dalla routine
  d) Questo legge un valore dal BANCO 0 della RAM ($0-$0FFF, $1000-$FFFF RAM) e lo pone nel registro C.

I. a) Funzione 1 dell'Utente: Scrive la funzione nel Banco 0 della RAM
  b) A=1 (numero della funzione)
  DE=indirizzo a 16 bit dove capita l'operazione di scrittura
  C=Valore da scrivere nello indirizzo del Banco 0
  c) A=0 al RITorno dalla routine se la scrittura è corretta
  A=-1 ($FF) al RITorno dalla routine se la scrittura non è corretta
  d) Questa funzione scrive il valore nel registro C nel banco 0 della RAM specificato dall'indirizzo nella coppia di registri DE. Un errore viene lampeggiato se viene tentata una scrittura in ROM ($0- $DFFFH) o nella pagina in alto della memoria ($FF00-$FFFF).

II. a) Funzione 2 dell'Utente: Funzione di Scansione dei Tasti
b) A=2 (numero della funzione)
c) B = -1 ($FF) se non viene premuto nessun tasto
B = posizione della matrice nella tabella delle matrici della tastiera (8se premuto)
A = contenuti (valore del carattere) della posizione della matrice se A è maiuscolo o minuscolo o contiene il tasto **CONTROL.**
HL = Indirizzo (puntatore) dove A (contenuto della posizione della matrice) si trova nella memoria
Indirizzo = Inizio Tabella + (4 * B) + C (bit 1 e 0)
C = Ritorna l'informazione del codice di controllo, dove ogni bit ha un significato specifico come segue:

Bit 1	& 0	
0	0	= minuscolo
0	1	= maiuscolo
1	0	= spostato
1	1	= tasto di controllo

C=		
	Bit 2	1 = tasto di controllo giù, 0 = su
	Bit 3	Non implementato
	Bit 4	1 = tasto shift destro giù,0 = su
	Bit 5	1 = tasto C è attivo (non necessar. giù)
	Bit 6	Non implementato
	Bit 7	1 = tasto shift sinistro giù, 0 = su

d) La funzione di scansione dei tasti permette all'utente di oltrepassare il normale procedimento di I/O BIOS della tastiera e di cercare un tasto particolare o una sequenza di tasti premuti.
e) Informazione Ulteriore-Indirizzi Importanti:
$FD09 = Puntatore dell'Inizio della Tabella (primo byte basso)

Ogni carattere ASCII ha quattro definizioni codificate. Ogni tasto ha un codice definito per quanto segue:

a) minuscolo
b) maiuscolo
c) tasto e carattere traslati
d) tasto e carattere di controllo

Queste quattro definizioni sono etichettate nelle colonne della tabella ASCII.

```
                     ASCII$TBL    A  B  C  D

1292  7F7F7F16       DB   7FH,7FH,7FH,16H      ; INS DEL posizione matrice 0
1296  0D0D0D0D       DB   0DH,0DH,0DH,0DH      ; RETURN
129A  06060101       DB   06H,06H,01H,01H      ; LF RT
129E  86868787       DB   86H,86H,87H,87H      ; F7 F8
12A2  80808181       DB   80H,80H,81H,81H      ; F1 F2
12A6  82828383       DB   82H,82H,83H,83H      ; F3 F4
12AA  84848585       DB   84H,84H,85H,85H      ; F5 F6
12AE  1717171A       DB   17H,17H,17H,1AH      ; UP DOWN

12B2  333323A2       DB   33H,33H,23H,0A2H     ; 3 #
12B6  77575717       DB   77H,57H,57H,17H      ; W
12BA  61414101       DB   61H,41H,41H,01H      ; A
12BE  343424A3       DB   34H,34H,24H,0A3H     ; 4$
12C2  7A5A5A1A       DB   7AH,5AH,5AH,1AH      ; Z
12C6  73535313       DB   73H,53H,53H,13H      ; S
12CA  65454505       DB   65H,45H,45H,05H      ; E
12CE  00000000       DB   00H,00H,00H,00H      ; (LF SHIFT)

12D2  353525A4       DB   35H,35H,25H,0A4H     ; 5 %
12D6  72525212       DB   72H,52H,52H,12H      ; R
12DA  64444404       DB   64H,44H,44H,04H      ; D
12DE  363626A5       DB   36H,36H,26H,0A5H     ; 6 &
12E2  63434303       DB   63H,43H,43H,03H      ; C
12E6  66464606       DB   66H,46H,46H,06H      ; F
12EA  74545414       DB   74H,54H,54H,14H      ; T
12EE  78585818       DB   78H,58H,58H,18H      ; X

12F2  373727A6       DB   37H,37H,27H,0A6H     ; 7 '
12F6  79595919       DB   79H,59H,59H,19H      ; Y
12FA  67474707       DB   67H,47H,47H,07H      ; G
12FE  383828A7       DB   38H,38H,28H,0A7H     ; 8 (
1302  62424202       DB   62H,42H,42H,02H      ; B
1306  68484808       DB   68H,48H,48H,08H      ; H
130A  75555515       DB   75H,55H,55H,15H      ; U
130E  76565616       DB   76H,56H,56H,16H      ; V

1312  39392900       DB   39H,39H,29H,00H      ; 9 )
1316  69494909       DB   69H,49H,49H,09H      ; I
131A  6A4A4A0A       DB   6AH,4AH,4AH,0AH      ; J
131E  30303000       DB   30H,30H,30H,00H      ; 0

1322  6D4D4D0D       DB   6DH,4DH,4DH,0DH      ; M
1326  6B4B4B0B       DB   6BH,4BH,4BH,0BH      ; K
132A  6F4F4F0F       DB   6FH,4FH,4FH,0FH      ; O
132E  6E4E4E0E       DB   6EH,4EH,4EH,0EH      ; N

1332  2B2B2B00       DB   2BH,2BH,2BH,00H      ; +
1336  70505010       DB   70H,50H,50H,10H      ; P
133A  6C4C4C0C       DB   6CH,4CH,4CH,0CH      ; L
133E  2D2D2D00       DB   2DH,2DH,2DH,00H      ; -
1342  2E2E3E00       DB   2EH,2EH,3EH,00H      ; , <
```

```
                    ASCII$TBL   A  B  C  D

1346  3A3A5B7B         DB   3AH,3AH,5BH,7BH      ; : [ {
134A  40404000         DB   40H,40H,40H,00H      ; @
134E  2C2C3C00         DB   2CH,2CH,3CH,00H      ; , <

1352  23232360         DB   23H,23H,23H,60H      ; POUND
1356  2A2A2A00         DB   2AH,2AH,2AH,00H      ; *
135A  3B3B5070         DB   3BH,3BH,5DH,7DH      ; ; ] }
135E  000000F5         DB   00H,00H,00H,0F5H     ; CLEAR/HOME
1362  00000000         DB   00H,00H,00H,00H      ; (RT SHIFT)
1366  3D3D3D7D         DB   3DH,3DH,3DH,7EH      ; =
136A  5E5E7C7C         DB   5EH,5EH,7CH,7CH      ; PI
136E  2F2F3F5C         DB   2FH,2FH,3FH,5CH      ; /?\

1372  3131210A         DB   31H,31H,21H,0A1H     ; 1
1376  5F5F5F7F         DB   5FH,5FH,5FH,7FH      ; <-
137A  09153000         DB   09H,15H,30H,00H      ; (CONTROL) SOUND1
                                                 SOUND 2
137E  323222A1         DB   32H,32H,22H,0A1H     ; 2"
1382  20202000         DB   20H,20H,20H,00H      ; SPACE
1386  21200000  21H,20H,00   ; (COMMODORE)
                      H,00H  SOUND3
138A  71515111         DB   71H,51H,51H,11H      ; Q
138E  000000F0         DB   00H,00H,00H,0F0H     ; RUN STOP
1392  9F9F9F9F         DB   9FH,9FH,9FH,9FH      ; /HELP/
1396  383838B7         DB   38H,38H,38H,0B7H     ; /8/
139A  353535B4         DB   35H,35H,35H,0B4H     ; /5/
139E  09090900         DB   09H,09H,09H,00H      ; /TAB/
13A2  323232B1         DB   32H,32H,32H,0B1H     ; /2/
13A6  343434B3         DB   34H,34H,34H,0B3H     ; /4/
13AA  373737B6         DB   37H,37H,37H,0B6H     ; /7/
13AE  313131B0         DB   31H,31H,31H,0B0H     ; /1/

13B2  1B1B1B00         DB   1BH,1BH,1BH,00H      ; /ESC/
13B6  2B2B2B00         DB   2BH,2BH,2BH,00H      ; /+/
13BA  2D2D2D00         DB   2DH,2DH,2DH,00H      ; /-/
13BE  0A0A0A0A         DB   0AH,0AH,0AH,0AH      ; /LINE FEED/
13C2  0D0D0DFF         DB   0DH,0DH,0DH,0FFH     ; /ENTR/
13C6  363636B5         DB   36H,36H,36H,0B5H     ; /6/
13CA  39393900         DB   39H,39H,39H,00H      ; /9/
13CE  333333B2         DB   33H,33H,33H,0B2H     ; /3/
13D2  00000000         DB   00H,00H,00H,00H      ; /ALT/
13D6  30303000         DB   30H,30H,30H,00H      ; /0/
13DA  2E2E2E00         DB   2EH,2EH,2EH,00H      ; /./
13DE  05050512         DB   05H,05H,05H,12H      ; /UP/
13E2  18181803         DB   18H,18H,18H,03H      ; /DN/
13E6  1313138D         DB   13H,13H,13H,08DH     ; /LF/
13EA  0404048E         DB   04H,04H,04H,08EH     ; /RT/
13EE  F1F1F1F2         DB   0F1H,0F1H,0F1H,0F2H  ; /NO SCROLL/

          ;
```

```
                ;       TABELLA LOGICA COLORE (USATA CON CHAR ESC ESC ESC)
                ;                             (DOVE CHAR VA DA 50H A 7FH)
                    ;

13F2  00112233           DB   000H,011H,022H,033H
13F6  44556677           DB   044H,055H,066H,077H
13FA  8899AABB           DB   088H,099H,0AAH,0BBH
13FE  CCDDEEFF           DB   0CCH,0DDH,0EEH,0FFH
```

III. a) Funzione 3 dell'Utente: Esegue una funzione ROM Z80
b) A = 3
L = numero della sottofunzione
Possono essere necessari ulteriori parametri di input
c) La maggior parte delle funzioni ROM non mette in output i valori; invece essi eseguono un'azione, che in questa funz. dell'utente è assunta essere l'output.
d) Questa funzione esegue una funzione ROM Z80. Queste routine ROM eseguono principalm. routine di gestione dello schermo. Tutti i numeri delle sottofunz. sono numeri pari. Le prime 80 sottofunzioni sono routine di gestione dello schermo per la visualizzazione a 40 ed 80 colonne. Qui viene listata solo la sottofunzione ad 80 colonne. Aggiungete 2 al numero della sottofunzione ad 80 colonne per ottenere il numero corrispond. della sottofunzione a 40 colonne.

III. 0a) Sottofunzione 0: Scrittura del Carattere
b) wr$char
c) Registro D = carattere per scrivere il cursore di auto avanzamento nella posizione successiva
d) ——

III. 4a) Sottofunzione 4: Posizione del Cursore
b) cursor$pos
c) Registro D = valore di riga
Registro E = valore di colonna
d) Questa sottofunzione pone la posizione corrente del cursore nello schermo ad 80 colonne.

III. 8a) Sottofunzione 8: Cursore su di una posizione
b) cursor$up
c) non sono ritornati valori
d) Questa sottofunzione muove il cursore in su di una riga nello schermo corrente (40 o 80), ma non oltre la cima dello schermo

III. 12a) Sottofunzione 12: Cursore giù di una riga
b) cursor$down
c) non sono ritornati valori
d) Questa sottofunzione muove il cursore giù di una riga nello schermo corrente (40 o 80). Lo schermo scrolla in giù se non c'è la linea in basso.

III. 14a) Sottofunzione 14: Non Implementata
b) ——
c) ——

III. 16a) Sottofunzione 16: Cursore a sinistra di una colonna
b) cursor$left
c) non sono ritornati valori
d) Questa sottofunzione muove il cursore a sinistra di una colonna, ma non oltre il margine sinistro.

III. 20a) Sottofunzione 20: Cursore a destra di una colonna
b) cursor$rt
c) non sono ritornati valori.
d) Questa sottofunzione muove il cursore a destra di una colonna, ma non oltre il margine destro dello schermo.

III. 24a) Sottofunzione 24: Esegue un ritorno carrello
b) do$cr
c) non sono ritornati valori.
d) Questa sottofunzione esegue un ritorno carrello e pone il cursore nel margine sinistro.

III. 28a) Sottofunzione 28: Azzera fino alla fine della linea
b) CEL
c) non sono ritornati valori.
d) Questa sottofunzione azzera la riga del cursore iniziando dove il cursore è correntem. localizzato e terminando alla fine della linea.

III. 32a) Sottofunzione 32: Azzera fino alla fine dello schermo
b) CES
c) non sono ritornati valori.
d) Questa sottofunzione azzera lo schermo partendo da dove il cursore è localizzato correntemente fino alla fine dello schermo.

III. 36a) Sottofunzione 36: Inserimento di carattere
b) char$ins
c) non sono ritornati valori.
d) Questa sottofunzione inserisce un carattere nella posizione corrente del cursore, l'ultimo carattere della linea viene perduto.

III. 40a) Sottofunzione 40: cancella carattere
b) char$del
c) non sono ritornati valori.
d) Questa sottofunzione cancella un carattere nella posizione corrente del cursore e pone uno spazio nell'ultima posizione della linea.

III. 44a) Sottofunzione 44: Inserimento di una linea
b) line$ins
c) non sono ritornati valori.
d) Questa sottofunzione inserisce una linea di spazi nella riga corrente del cursore, la riga corrente del cursore viene spostata in giù di una riga e l'ultima linea (24) viene persa.

III. 48a) Sottofunzione 48: azzera linea
b) line$del
c) non sono ritornati valori.
d) Questa sottofunzione cancella una linea alla riga carattere segnata dalla posizione del cursore corrente. La linea in basso (24) viene riempita di spazi.

III. 50a) Sottofunzione 50: Non implementata
b) ——
c) ——
d) ——

III. 52a) Sottofunzione 52: pone il colore del cursore
b) colore. Il registro B ritorna il valore del codice colore
c) non sono ritornati valori.
d) Questa funzione pone il codice del colore del cursore corrente con il valore in colore.

III. 56a) Sottofunzione 56: pone gli attributi a 80 colonne
b) attr
B = bit da porre/cancellare
C = valore del bit
c) non sono ritornati valori.
d) Questa funzione abilita/ disabilita gli attributi dell'8563 per lo schermo ad 80 colonne. Gli attributi sono funzioni extra schermo come lo sfondo, R,G,B e I, lampeggio, sottolineatura, video inverso e l'insieme di caratteri alternativi. La funzione a 40 colonne (sottofunz. 58) controlla solo il video inverso.

III. 60a) Sottofunzione 60: legge attributo del carattere (80)
b) rdchratr
D = riga carattere 8563
E = colonna carattere 8563
c) D = riga carattere 8563
L = colonna carattere 8563
B = valore del carattere
C =forma del bit attributo della locazione selez. del car.
d) Questa funzione legge il byte attributo della riga e colonna del car. selezionate nello schermo 8563 e ritorna il vero colore RGBI. La sottofunzione a 40 colonne corrisp. (62) controlla solo video inverso.

III. 64a) Sottofunzione 64: scrive attributo del carattere (80)
b) wrchratr
B = valore del carattere
C =attributo
c) non sono ritornati valori.
d) Questa funzione scrive un byte attributo del valore del carattere selezionato sullo schermo 8563.

III. 68a) Sottofunzione 68: legge colore
b) rd$color
c) A = colore del carattere
B = colore di fondo
C = colore del bordo (solo 40 colonne)
D = attributo 8563
d) Questa funzione legge il colore corrente delle sorgenti di colore 8563. La sottofunzione 70 ritorna le sorgenti VIC del colore.

III. 80a) Sottofunzione 80: converte il record (solo GCR)
b) @trk
c) VICtrk, VICSEC
d) Questa funzione ritorna la traccia fisica ed il settore dal drive, compreso lo skew (disallinemento), data la traccia logica.

III. 82a) Sottofunzione 82: controlla il codice CBM da dischetto (solo GCR)
b) check$CBM
c) flag zero = 1 se CBM (modo disco C128) è presente nel drive
flag zero = 0 se CP/M 2.2 (disco CP/M C64) è presente nel drive
A = 0 se a singola faccia
A = FF se a doppia faccia
d) Questa funzione rileva che tipo di dischetto GCR è nel drive. Legge i dati da t = 1, s = 0 nel buffer a $FE00.

III. 84a) Sottofunzione 84: funzione campanello
b) Suono 1, Suono 2, Suono 3 ($FD10, $FD12, $FD14, rispettiv)
c) Emette un suono di campanello
d) Questa funzione emette un suono di campanello dal chip SID. Le sottofunzioni 86-95 non sono definite.

III. 96a) Sottofunzione 96: traccia 40
b) trk$40
c) @off40
d) Questa funzione calcola la posizione logica (offset) del cursore nello schermo fisico ad 80 colonne. La variabile @off40 è calcolata sui limiti di 8 caratteri ed è usata nella seguente sottofun.

III. 98a) Sottofunzione 98: pone la posizione del cursore
b) setcur40
c) non sono ritornati valori.
d) Questa funzione chiama la Traccia 40 per controllare l'offset tra lo schermo logico (finestra) e quello fisico (40 colonne). Mantiene il cursore all'interno della finestra logica a 40 colonne sullo schermo virtuale ad 80 colonne che ha subito lo scroll per l'output dello schermo VIC.

III. 100a) Sottofunzione 100: dipinge una linea
b) line$paint
c) non sono ritornati valori.
d) Questa funzione aggiorna la linea corrente di caratteri solo se @off40 e old $offset sono uguali. Se old $offset = -1, viene chiamata la TRACCIA 40. Se @off40 e old $offset non sono uguali, è chiamato Dipingi Schermo per lo scroll dello schermo logico a 40 colonne (finestra) sullo schermo virtuale ad 80 colonne.

III. 102a) Sottofunzione 102: dipingi schermo
b) screen$paint
c) non sono ritornati valori.
d) Questa funzione aggiorna (fa lo scroll) lo schermo logico a 40 colonne (finestra) attraverso lo schermo virtuale VIC ad 80 colonne basato sul valore di @off40. Questa e le tre sottofunzioni precedenti (96,98,100) sono intrecciate per rendere la visualizzazione dello schermo logico a 40 col che ha subito lo scroll più veloce possibile. Normalmente dovreste chiamare solo Poni Posizione del Cursore o Dipingi Linea, poichè esse chiamano le altre routine connesse se necessario. Consultate il listato del file CXROM.ASM nei dischetti DRI che hanno la documentazione delle fonti dei listati.

III. 104a) Sottofunzione 104: stampa messaggio (BOTH) (entrambi)
b) prtmsgboth
c) non sono ritornati valori.

d) Questa funzione stampa gli output contemporan. ad entrambi gli schermi, visualizzando le stringhe puntate dal valore in alto nello stack. Pone l'indirizzo della stringa nello stack e termina la stringa con uno zero. L'esecuzione riprende con il byte seguente lo zero. Funziona come una "Stampa in programma di linea".

III. 106a) Sottofunzione 106: stampa messaggio (BOTH) (entrambi)
b) prtdeboth
c) non sono ritornati valori.
d) Questa funzione opera come la precedente, tranne che l'indirizzo di partenza della stringa viene preso da DE e che l'esecuzione riprende con l'indirizzo di ritorno dallo stack.

III. 108a) Sottofunzione 108: messaggio nascosto
b) update$it
c) non sono riportati valori.
d) Questa funzione visualizza i messaggi nascosti. La sottofunzione 110 non è implementata.

III. 112a) Sottofunzione 112: conversione da ASCII in PET ASCII
b) ASCII$pet, registro B = carattere ASCII ($20—$7F)
c) Registro A = carattere convertito in PET ASCII.
d) Questa funzione esegue una conversione standard da ASCII in PET ASCII sui caratteri stampati nello schermo (da qualsiasi unità di input). I codici di controllo non sono tradotti.

III. 114a) Sottofunzione 114: pone il cursore 40 colonne nell'indirizzo specificato
b) curadr40
c) HL = indirizzo del cursore su schermo
DE = indirizzo di partenza della linea (riga) del cursore
BC = # dei caratteri a fine riga (<80, cursore non di conteggio)
d) Questa funzione pone il cursore all'indirizzo specificato in HL (nel banco 0 della RAM). Questo indiriz. è dello schermo logico, non di quello virtuale.

III. 116a) Sottofunzione 116: pone il cursore ad 80 colonne nell'indirizzo specificato
b) curadr80
c) HL = indirizzo del cursore su schermo
DE = indirizzo di partenza della linea (riga) del cursore
BC = # dei caratteri alla fine della linea (<80, cursore non di conteggio)
d) Questa funzione pone il cursore all'indirizzo specif in HL (nella RAM 8563).

III. 118a) Sottofunzione 118: cerca il colore
b) look$color, B = codice colore ($30—$3F), C = max. Valore del codice colore.
c) HL = puntatore della tabella del colore logico (nybble inferiore = 80, nybble alto = 40)
B = $0 (Carattere colore)
$10 (Colore di fondo)
$20 (Colore dei margini)
d) Questa funzione pone i colori dello schermo 8563 nei colori dello schermo VIC. La sottofunzione 120 non è definita.

III. 122a) Sottofunzione 122: riempi blocco
b) blk$fill, pone l'indirizzo di partenza nello stack (8563)
BC = # dei byte da riempire
D = carattere riempimento
E = attributo
c) non sono ritornati valori.
d) Questa funzione riempie un blocco di 256 byte con i dati specificati nel registro D.

III. 124a) Sottofunzione 124: muovi blocco
b) blk$move, pone l'indirizzo sorgente (nella RAM 8563) nello stack
DE = indirizzo di destinazione (nella RAM 8563)
BC = conto
c) non sono ritornati valori.
d) Questa funzione muove i blocchi di 256 byte da un blocco di memoria ad un altro nella RAM 8563.

NOTA: Le sottofunzioni 122, 124 e 126 devono essere chiamate direttamente dallo Z80. Le routine di esempio fornite in questa parte per le sottofunzioni 122, 124 e 126 non funzioneranno. Esse di solito non sono accessibili da parte dell'utente e devono girare nel BANCO 0.

III. 126a) Sottofunzione 126: installazione carattere
b) chr$inst, stack = indirizzo della RAM 8563 per installare la definizione del carattere
DE = indirizzo della memoria del sistema (RAM C128 (banco 0)) della nuova definizione del carattere (8 byte per carattere).
B = numero dei caratteri consecutivi
c) No valori
d) Installa un carattere definito dell'utente nella RAM 8563.

IV. a) Funzione 4 dell'Utente: Funzioni BIOS 8502
b) A = 4 (numero della funzione)
L = numero della sottofunzione (da −1 a 11)
Possono essere richiesti ulteriori parametri di input
c) Gli output dipendono da ogni sottofunzione
d) La funzione 4 dell'utente vi permette di chiamare le funzioni input/output 8502 che sono eseguite dal processore 8502. Queste funz. non sono parte del sistema CP/M standard e sono completamente hardware dipendenti. Questa funzione vi abilita ad andare avanti e indietro tra i processori Z80 e 8502.
e) Ogni sottofunzione è trattata in dettaglio usando le convenzioni definite.

IV. -1.a) Sottofunzione -1: fa di nuovo il boot dell'hardware del C128
b) A = 4
L = -1
c) nessuno
d) Questa sottofunzione fa di nuovo il boot dell'hardware del Commodore 128 quando il valore del registro L è uguale a -1 ($FF). Questa sottofunzione non è usata normalmente. Esegue le stesse azioni come se premeste reset.

IV. 0.a) Sottofunzione 0: inizializza il BIOS 8502
c) A = 4
L = 0
c1) Predispone il vettore di interruzione $0314-$0319 nel vettore dell'handler di interruzione 8502
c2) Copia il vettore di interruz. ROM nella RAM
c3) Predispone le variabili PAL e NTSC (SYSFREQ)
c4) Chiude tutti i canali aperti
d) Questa sottofunzione inizializ. le variabili del sistema 8502, i procedimenti di interruzione e le frequenze del sistema in modo che lo Z80 e l'8502 possano comunicare in avanti e indietro. Il controllo del processore viene dato o allo Z80 o all'8502 in un momento particolare. I processori non possono girare simultaneamente
IV. 1.a) Sottofunzione 1: Lettura 1541 (solo formato GCR)
b) A=4
L=1
VICTRACK=(1-35)—Variabile del numero di traccia su disco
VICSECTOR=(0-21)—Variabile del numero di settore su disco
VICDRV=—Variabile del numero di unità del dischetto dove:

NYBBLE INFERIORE DI VICDRV				EQUIVALENTE APERTO
VALORE DEL BIT		NUMERO DI UNITÀ	DRIVE	ISTRUZIONE IN BASIC
0001	=	8	0	OPEN 8,11,15
0010	=	9	0	OPEN 9,12,16
0100	=	10	0	OPEN 10,13,17
1000	=	11	0	OPEN 11,14,18

Valori del Nybble Inferiore di VICDRV

c) VICDATA = 11 ($0B) se il disco nel drive è stato modificato
VICDATA = 13 ($0D) se capita un errore di lettura/scrittura o di canale
VICDATA = 0 se la lettura è corretta
VICDATA = 15 ($0F) se l'unità non è presente
d) Questa funzione legge traccia e settore particolari del dischetto nel drive come specificato da VICTRACK, VICSECTOR e VICDRV rispettiv. Il dato viene letto nel buffer in $FE00H. Il valore ritornato nella variabile VICDATA dipende dalle condizioni descritte in c.
e) Informazione Ulteriore: Questa sottofunzione assume che entrambi i canali dato e comando siano stati aperti in precedenza. Capiterà un errore se questa routine viene chiamata per leggere da un 1571.
IV. 2.a) Sottofunzione 2: Scrittura 1541 (solo formato GCR)
b) A=4
L=2
VICTRACK = lo stesso della lettura 1541
VICSECTOR = lo stesso della lettura 1541
VICDRV = lo stesso della lettura 1541
c) VICDATA = lo stesso della lettura 1541

d) Questa sottofunzione scrive i dati nella traccia e settore specificati del dischetto nel drive come specificato da VICTRACK, VICSECTOR e VICDRV rispettivamente. Il valore ritornato in VICDATA dipende dalle condizioni descritte per gli output in c. Consultate la sottofunzione 1 per i dettagli

e) Informazione Ulteriore: Questa sottofunzione assume che entrambi i canali dato e comando siano stati aperti prima. Il dato viene scritto dal buffer in $FE00H.

IV. 3.a) Sottofunzione 3: Predisposiz della Lettura 1571 (formati MFM o GCR)

b) A=4
L=3
*VICTRACK = (1-35)—Variabile per il num. di traccia su dischetto
*VICSECTOR = (0-21)—Variabile per il num. di settore su dischetto
*VICDRV = —Variabile per il num. di unità del drive (Consultate la tabella VICDRV a pag. precedente).
* = Gli intervalli si adattano solo al formato GCR. Gli intervalli sono diversi per i dischetti MFM dipendendo da chi li costruisce.
VIC$COUNT = Numero dei settori da leggere (sulla traccia)

c) VICDATA = 11 ($0B) se il disco nel drive è stato modificato
VICDATA = 12 ($0C) se il drive non è un drive veloce (1571)
VICDATA = 13 ($0D) se si verifica un errore di canale
VICDATA = 15 ($0F) se l'unità non è presente
Se FAST coincide (ANDed) con VICDRV = 0 il drive significativo è un 1541
Se FAST coincide con VICDRV = 1 il drive significativo è un 1571

d) Questa sottofunzione predispone il drive 1571 per un'operazione di lettura. Tuttavia, il trasferimento dei dati non è eseguito da BIOS 8502. I dati sono trasferiti dallo Z80.

e) Informazione Ulteriore: Per accedere al retro di un disco MFM ponete il bit 7 ($80) di VICSECTOR. Per i formati MFM, un dash (tratto) tra la traccia ed il settore nella finestra visualizzata significa che il drive ha accesso al retro del dischetto. Ciò viene eseguito di solito dal BIOS.

IV. 4.a) Sottofunzione 4: Predisposiz della Scrittura 1571 (formati MFM o GCR)

b) A=4
L=4
* VICTRACK = (1-35)—Variabile per il num. di traccia su dischetto
* VICSECTOR = (0-21)—Variabile per il num. di settore su dischetto
VICDRV = —Variabile per il numero di unità del drive. Consultate la tabella VICDRV della pagina precedente.
VIC$COUNT = Numero dei settori da leggere
* = Intervalli che si adattano solo al formato GCR. Gli intervalli sono diversi per i dischi MFM dipendendo dal costruttore.

c) VICDATA = 11 ($0B) se il disco nel drive è stato modificato
VICDATA = 12 ($0C) se il drive non è un drive veloce (1571)
VICDATA = 13 ($0D) se si verifica un errore di canale
VICDATA = 15 ($0F) se l'unità non è presente
Se FAST coincide con VICDRV = 0 il drive significativo è un 1541
Se FAST coincide con VICDRV = 1 il drive significativo è un 1571

d) Questa sottofunzione predispone il drive 1571 per un'operazione di scrittura. Tuttavia, il dato non è eseguito dal BIOS 8502. Il dato è trasferito dallo Z80.

e) Informazione Ulteriore:Ecco come l'utente dovrebbe selezionare tra un Drive 1541 e 1571. Per accedere al retro di un disco MFM ponete il bit 7 di VICSECTOR. Per eseguire un'operazione di scrittura, l'utente dovrà fare ciò in applicazione.

IV. 5.a) Sottofunzione 5: Interroga il Drive 1541 o 1571

b) A=4
L=5
VICDRV = (8-11)—Variabile del numero di unità del drive

c) VICDATA =i quattro bit più bassi ritornano lo stato per la lettura /scrittura veloce (uguale ai codici di errore del Disco VICDATA precedente che sono listati nella tabella di Stato degli Errori su Disco della pag. seguente).
= i quattro bit più alti ritornano la misura del settore per i dischetti MFM

d) Questa sottofunzione interroga il drive per sapere la misura del settore del dischetto (MFM o GCR) e lo stato del drive. Inoltre, questa sottofunzione inizializza la variabile FAST, chiude e riapre il canale per il drive corrispondente ed azzera lo stato del drive.

IV. 6.a) Sottofunzione 6: Domanda al Dischetto

b) A=4
L=6
VICTRACK = (1-35)—Variabile per il numero di traccia su disco
VICSECTOR = (0-21)—Variabile per numero di settore su disco
VICDRV = —Variabile per il numero di unità del drive

c) Se senza errori VICDATA = i quattro bit più bassi ritornano lo stato per la lettura /scrittura veloce = i quattro bit più alti ritornano la misura del settore per i dischetti MFM e la funzione dà in input 6 byte in un buffer di memoria che inizia alla locazione $FE00 (e finisce in $FEFF). Questi 6 byte sono definiti come:

1) $FE00-TRACK STATUS (nella traccia sotto)
2) $FE01-Numero di Settori
3) $FE02-Traccia Logica
4) $FE03-Numero Minimo di Settori (su questa traccia)
5) $FE04-Numero Massimo di Settori (su questa traccia)
6) $FE05-Intercalato Fisico

Se capita un errore:

VICDATA = 11 ($0B) se il disco nel drive è stato cambiato
VICDATA = 12 ($0C) se il drive non è un drive veloce (1571)
VICDATA = 13 ($0D) se si verifica un errore di canale
VICDATA = 15 ($0F) se non è presente l'unità

d) Questa sottofunzione domanda al dischetto e ritorna lo stato del dischetto e la misura del settore (se in formato MFM). Inoltre, il buffer localizzato tra $FE00 e $FEFF riceve 6 dati come descritto sopra.

Tabella dello Stato di Errore del Dischetto

MD DN I1 I2 D1 D2 D3 D4

MD .... Modo: 1 = MFM, 0 = GCR
DN .... Numero del Drive
I1,I2 .... Misura del settore
- a) 00 = 128 byte
- b) 01 = 256 byte
- c) 10 = 512 byte
- d) 11 = 1024 byte

D1-D4 .... Stato del Controllore

GCR

000x = Ok.
0010 = Settore non trovato.
0011 = No Sincronismo.
0100 = Blocco di dati non trovato.
0101 = Verifica per somma del blocco di dati.
0110 = Errore di Formato.
0111 = Errore di Verifica.
1000 = Errore di protezione della Scrittura.
1001 = Verifica per somma del blocco della Testata.
1010 = Il dato si estende nel blocco successivo.
1011 = Non corrispondenza del Disco ID /Cambio dischetto.
1100 = Il drive non è veloce (1571).
1101 = Errore di Canale.
1110 = Sintassi.
1111 = Drive non presente.

MFM

000x = Ok.
0010 = Settore non trovato.
0011 = No segno di indirizzo.
0100 = Non usato.
0101 = Errore del Dato CRC.
0110 = Errore di Formato.
0111 = Errore di Verifica.
1000 = Errore di protezione di Scrittura.
1001 = Errore di Testata CRC.
1010 = Non usata.
1011 = Cambiamento di dischetto.
1100 = Il drive non è veloce (1571).
1101 = Errore di Canale.
1110 = Sintassi.
1111 = Drive non presente.

Tabella dello Stato di Errore del Dischetto

IV. 7.a) Sottofunzione 7: Stampa i caratteri verso una stampante con bus seriale

b) A=4
L=7
VICDRV = numero della stampante (4 o 5)
VICTRACK = indirizzo secondario nel quale l'unità è aperta come
VICDATA = carattere da stampare nella stampante a bus seriale (se VICCOUNT = 0)

c) VICDATA = -1 ($FF) se l'unità non è presente

d) Questa sottofunzione emette i caratteri nella stampante a bus seriale precedentemente aperta.

e) Informazione Ulteriore: Se l'indirizzo secondario (che di solito è 7) è cambiato, l'unità è chiusa e riaperta con il nuovo indirizzo secondar. Se si verifica un errore del bus seriale oltre all'unità non presente, il canale viene chiuso riaperto e l'operazione originale viene ancora eseguita. Se VICCOUNT non è uguale a zero il dato è stampato dal buffer puntato da $FE00. Il numero di byte stampato è fornito in VICCOUNT.

IV. 8.a) Sottofunzione 8: Formatta un Dischetto 1541 o 1571

b) A=4
L=8
DRIVE#
FAST

c) VICDATA

d) Questa sottofunzione formatta un dischetto 1541 o 1571 nel drive appropriato. Se FAST è abilitato (FAST connesso con DRIVE# (non uguale a 0)), la lunghezza di un comando del dischetto è presa dalla locazione $FE00 del buffer della memoria. Il comando che inizia alla locazione $FE01 e che termina alla locazione specificata dalla lunghezza del comando in $FE00 è inviato al drive. Per esempio, se $FE00=$06, il comando nel buffer della memoria tra $FE01 e $FE06 è inviato al drive. Tutti i comandi sono preceduti da un "U0", così solo il comando del buffer della memoria deve essere sostituito.

IV. 9.a) Sottofunzione 9: Chiamata del l'utente al Codice della Routine 8502

b) A=4
L=9
VICOUNT = ($FD05) indirizzo del byte basso della routine 8502 (puntatore all'inizio della esecuzione della routine dell'utente)
VICDATA = ($FD06) indirizzo del byte alto della routine 8502

c) Solo output definiti dall'utente

d) Questa è la routine che vi permette di chiamare una subroutine 8502 in linguaggio macchina dal modo Z80. La routine 8502 codificata di solito è definita dall'utente. Deve fare le esecuzioni nel banco 0 della RAM con i registri di input/ output abilitati. Il valore MMU è $3E. Il controllo viene trasferito al processore 8502 con il KERNAL disabilitato. Se volete chiamare una routine KERNAL C128, dovete abilitare il KERNAL dopo aver trasferito il controllo all'8502. Prima di ritornare allo Z80, dovete ancora disabilitare il KERNAL. Quando il controllo passa dal processorre Z80 all'8502, lo Z80 è inattivo. Per ridare il controllo al processore Z80, ponete la solita istruzione RTS 8502 alla fine della vostra routine 8502 codificata ed il controllo viene ridato allo Z80.

e) Informazione Ulteriore: Una volta che il controllo è passato dallo Z80 all'8502, l'8502 gira alla velocità di 1 Mhz. Potete aumentare la velocità a 2 Mhz per accelerare il procedimento dalla parte 8502 del computer. Tuttavia, DOVETE RITORNARE ALLA VELOCITÀ DI 1 Mhz PRIMA DI TORNARE ALLO Z80 O IL SISTEMA SI IMPIANTERÀ. La natura della temporizzazione dei due processori fa sì che ciò accada. Se non ritornate a 1 Mhz la temporizzazione del clock è eliminata ed il sistema si impianta.

IV.10.a) Sottofunzione 10: Lettura della RAM del dischetto
b) A=4
L=10
c) I dati sono trasferiti per la espansione della RAM dalla RAM (BANCO 0)
d) Tutti i registri dell'espansione della RAM devono essere predisposti prima di chiamare questa routine.

IV.11.a) Sottofunzione 11: Scrittura nella RAM del dischetto
b) A=4
L=11
c) I dati sono trasferiti per la espansione della RAM dalla RAM (BANCO 0)
d) Tutti i registri dell'espansione della RAM devono essere predisposti prima di chiamare questa routine.

V. a) Funzione 5 dell'Utente: Legge la Colonna 40/80 dei Tasti
b) A=5 (numero della funzione)
c) Valore memorizzato in $D505 (registro del Modo di Configuraz. C128) Se il bit 7 è alto (1), il tasto 40/80 è su altrimenti 40/80 è giù.
d) Questa funzione ritorna il valore della locazione $D505, il registro del modo di configurazione. Solo il bit 7 è significativo coma notato sopra. Il tasto 40/80 non è nella tabella delle matrici della tastiera, così questa funzione è dedicata alla lettura della sua posizione.

VI. a) Le funzioni dalla 6 alla 254 non sono implementate.
b) Nessuno
c) HL = numero dei giorni (in binari) dal 1/1/78

VII. a) Funzione 255 dell'Utente: Data del Sistema
b) A=−1 ($FF)
c) HL=numero dei giorni (in binari) dal 1/1/78
d) Specificando A=−1, viene ritornata la data del sistema.

APPENDICE K - PARTE II

CHIAMATE DI CP/M BIOS, BIOS 8502 E DELLE FUNZIONI DELL'UTENTE IN LINGUAGGIO MACCHINA Z80

Il sistema CP/M del Commodore 128 vi permette di chiamare CP/M BIOS, BIOS 8502 e le funzioni CP/M dell'utente nei vostri programmi Z80 di linguaggio assembler. Tuttavia, per programmare in linguaggio assembler Z80, avete bisogno di un monitor assembler (assemblatore) o di linguaggio macchina. Molti assemblatori e monitor Z80 sono disponibili sul mercato; tuttavia, il sistema completo Digital Research CP/M Plus (3.0) ora disponibile sul Commodore 128 si compra con due assemblatori Z80, MAC e RMAC e la documentazione del CP/M Plus. Programmi e documentazione non sono inclusi nel pacchetto del Personal Computer Commodore 128.

Assumendo che voi abbiate gli assemblatori MAC e RMAC, potete ora entrare ed assemblare i programmi Z80 in linguaggio assembler. A questo punto, questa guida di consultazione deve dare per scontato qualcosa di sostanzialmente importante a proposito dei lettori e della loro conoscenza di programmazione in linguaggio di assemblaggio dello Z80. Come probabilmente concordate, questa guida di consultazione non potrebbe introdurre il linguaggio macchina dello Z80 e trattarlo completamente. Ciò va oltre lo scopo di questo libro. Le vostre librerie di fiducia offrono senza dubbio parecchi libri validi sulla programmazione dello Z80. Ora che sono state dette tutte le cose date per scontate, ecco come chiamare una funzione dell'utente dello Z80 o BIOS. Per prima cosa la richiesta di una funzione dell'utente.

CHIAMARE UNA FUNZIONE DELL'UTENTE NEL SISTEMA CP/M

Come avete visto nella Parte I di questa appendice, certe funzioni dell'utente avevano sottofunzioni, altre no. La funzione 4 dell'utente è la funzione di chiamata BIOS 8502. Le funzioni BIOS 8502 hanno 13 (–1-11) sottofunzioni che eseguono le routine di input e output del livello macchina 8502. Non confondete

il BIOS 8502 con il BIOS CP/M. Il BIOS CP/M è standard rispetto a qualsiasi sistema CP/M Plus senza tenere conto dell'hardware che lo fa girare. Ricordate che il CP/M è stato progettato per essere trasportato da un microcomputer all'altro. Ogni microcomputer diverso ha il proprio livello macchina di input e output che esso deve eseguire. Il BIOS 8502 è completamente hardware dipendente e girerà solo sul microprocessore 8502 del Commodore 128.

Un altro modo per capire la differenza tra i due tipi BIOS è quello di richiamare il vettore di salto CP/M della prima parte dell'Appendice K. I primi 30 salti (0-29) sono chiamate dirette delle routine BIOS CP/M. Il salto numero 30 è la chiamata della funzione dell'utente, della quale la funzione 4 dell'utente è il BIOS 8502. Il BIOS 8502 è un sottoinsieme delle funzioni dell'utente CP/M. La chiamata della funzione dell'utente è 1 delle 31 chiamate delle routine del sistema all'interno della tabella di salto del BIOS CP/M.

L'esempio seguente chiama la funzione 2 dell'utente, la funzione di scansione dei tasti. La routine di chiamata inizia al "waitkey" e la subroutine inizia a "user$fun".

```
            MVI useroffset,30
waitkey:
            MVI A,2              ;carica reg A con funz. numero 2
            CALL user$fun        ;chiama la subroutine user$fun
            INR B                ;incrementa B—test per -1 se non viene
                                   premuto un tasto
            JZ waitkey           ;salta al tasto wait se non ce n'è nessu-
                                   no
            DCR B                ;decrementa B se B non è uguale a 0
                                 ;questo ristabilisce il valore originale della
                                   matrice
             .
             .
             .
            Resto del
            Programma
             .
             .
user$fun
            PUSH H               ;pone la coppia di reg HL nello stack
            LHLD 1               ;carica l'indirizzo del vettore 1 di salto
                                   (WBOOT) in HL
            MVI L,useroffset *3  ;porta l'offset di 90 (30*3) a L
            LXTHL                ;scambia HL con la parte alta dello stack
            RET                  ;salta alla nuova locaz del puntatore HL
                                   della routine
```

Primo, il programma principale carica il numero (2) della funzione dell'utente nel registro A. Per chiamare una funzione dell'utente, ponete il parametro di input richiesto e il numero della funzione dell'utente nel registro A. Se una sottofunzione sta per essere chiamata, come per la funzione 4 dell'utente (BIOS 8502), il parametro di input per il numero della sottofunzione deve essere posto in L.

La seconda istruzione della routine principale è una CHIAMATA della subroutine user$fun. Tuttavia, in questo esempio della funzione dell'utente per la scansione dei tasti, il valore B ritornato è -1 se non vengono premuti tasti. Se questo è il caso, B viene incrementato a zero ed il programma principale salta a waitkey e fa la scansione della tastiera ancora una volta. Altrimenti, il resto del programma continua l'elaborazione.

Quando la subroutine viene chiamata, la prima istruzione salva la coppia di registri HL ed il puntatore dell'indirizzo in cima allo stack. L'istruzione successiva carica la locazione di memoria 1 (bassa) e 2 (alta) nella coppia di registri HL. Il byte alto punta il numero di pagina in cui si trovano i vettori di salto ed il byte basso è sempre 3 (Salto #1 vettore di boot caldo). Poi il numero di salto 3 viene caricato nel byte basso (L) della coppia di registri HL. Questo aumenta l'offset di 90 locazioni di memoria all'indirizzo di base della tabella di salto BIOS CP/M, che ora punta al numero di salto 30, USERF.

L'istruzione XTHL cambia la coppia di registri HL con la cima dello stack. Questo pone l'indirizzo calcolato in cima allo stack ed i valori di entrata di HL di nuovo in HL. Quando l'istruzione RETurn viene raggiunta dallo Z80, il controllo del programma viene passato al vettore USERF, all'entrata 30 della tabella di salto del CP/M BIOS. Quando la funzione è completata, il controllo ritorna all'istruzione che segue immediatamente l'istruzione CALL User$Fun (INR B).

Perchè le routine siano chiamate in modo adeguato, i parametri giusti di input richiesti devono essere posti nei registri adeguati. Il numero della funzione dell'utente deve essere posto nel registro A, il numero della sottofunzione è posto in L, se ce ne sono. Gli input ulteriori devono essere posti nel registro o variabile giusti prima di chiamare la funzione dell'utente.

L'esempio di cui sopra chiama la funzione 2 dell'utente, la funzione di scansione dei tasti. Per chiamare qualsiasi altra funzione dell'utente, caricate il numero della sottofunzione nel registro A. Per chiamare una sottofunzione, caricate il registro L con un numero di sottofunzione all'inizio del programma principale (di chiamata) come segue:

```
MVI L, subfun
```

Il modo in cui è scritto questo programma, il valore in L, che caricherete all'inizio del programma con l'istruzione di cui sopra, viene posto nuovamente in L dallo stack quando XTHL viene raggiunto alla fine della subroutine. Usate questo esempio come dima quando chiamate altre sottofunzioni. Assicuratevi che gli input adeguati siano presenti nelle locazioni giuste prima di chiamare la funzione dell'utente.

CHIAMATA DI UNA ROUTINE CP/M BIOS

Fare una chiamata diretta BIOS è diverso dalla chiamata di una funzione dell'utente. Le chiamate della funzione dell'utente inserisce sempre il numero di salto 30 nella tabella di salto del CP/M BIOS. Una chiamata diretta BIOS inse-

risce uno qualsiasi dei primi 30 vettori di salto (0-29). Queste sono le routine di input ed output che sono parte di qualsiasi sistema CP/M di qualsiasi microcomputer. Il metodo standard per chiamare una routine CP/M BIOS è con la funzione 50 BIOS. Questa funzione gestisce l'immissione nei due banchi a 64K della RAM del C128.

Potreste chiamare una routine CP/M BIOS simile a quella del primo esempio, ma c'è una limitazione. Con il metodo di chiamata della funzione dell'utente, siete in grado solo di chiamare le routine BIOS che risiedono nel banco 1 della RAM. TPA risiede nel banco 1, mentre il codice della funzione Z80 è memorizzato nel banco 0 della RAM. Se tentate di chiamare una routine diretta BIOS nel banco 0, il sistema si impianterà perchè il banco 1 è in contesto. Ecco perchè è importante fare chiamate dirette CP/M BIOS attraverso la funzione 50 dell'utente. Consultate la documentazione del Digital Research CP/M Plus per ulteriori dettagli sulla funzione 50 dell'utente.

Ecco un esempio di programma che illustra come chiamare una funzione CP/M BIOS.

```
principale
          MVI C,50          ;memorizza la funzione num 50 nel reg C
          LXI D,bios$pb     ;carica subito un puntatore a 16 bit in DE
          CALL 5            ;chiamata standard BIOS
            .
            .
            .
          Resto del Program.
            .
            .
            .
          RET
bios$pb:                    ;tabella delle variab. BIOS
          db FUNNUM         ;num. della variabile della funzione BIOS
          db AREG           ;memorizzaz. temporanea nel registro A
          dw BCREG          ;memorizzaz. temporanea nella coppia di
                            reg BC
          dw DEREG          ;memorizzaz. temporanea nella coppia di
                            reg DE
          dw HLREG          ;memorizzaz. temporanea nella coppia di
                            reg HL
```

La prima istruzione della routine principale memorizza il numero della funzione 50 nel registro C. Il sistema si aspetta che il parametro C di input sia il numero della funzione BIOS. L'istruzione successiva carica l'indirizzo della tabella delle variabili BIOS bios$pb nella coppia di registri DE. La terza istruzione chiama la funzione 50 BDOS per mezzo del vettore di chiamata BDOS, il vettore standard BIOS per mezzo del quale sono chiamate tutte le funzioni BIOS dirette. La funzione BDOS 50 gestisce l'inserimento ed il disinserimento nei banchi 0 e 1 della RAM. Questo è il modo consigliato per chiamare direttamente una funzione CP/M BIOS oltre quello del primo esempio che è stato creato per prima cosa per chiamare le funzioni dell'utente. La tabella delle variabili "bios$pb" contiene i

parametri di input richiesti necessari per la funzione 50 BDOS. "db" è un byte di memorizzazione (come il .byte nell'8502) mentre "dw" è una parola di 16 bit (come .parola nell'8502).

Questa appendice è la sola sezione del libro che incontra la barriera della programmazione dello Z80. Almeno vi dirige nella direzione giusta e vi dà "l'aggancio" per le routine di livello macchina del sistema CP/M.

FORMATI DEL DISCHETTO MFM

L'abbreviazione MFM sta per Modified Frequency Modulation (Modulazione di Frequenza Modificata). Questo tipo di formato di dischetto è variabile e programmabile secondo le specifiche di costruzione di ciascun computer.

Il sistema CP/M Commodore 128 sopporta quattro formati standard MFM del dischetto. Questi quattro formati costituiscono la maggioranza dei formati software CP/M disponibili sul mercato. Questo non significa che il sistema CP/M Commodore 128 può leggere tutti i formati CP/M dei dischetti dell'universo; tuttavia, la maggioranza del software CP/M può girare sul Commodore 128, se l'applicazione specifica non è dipendente dall'hardware.

I quattro formati principali che sono sopportati sono i seguenti:

- a) Epson QX10 (a singola faccia)
- a1) Epson QX10 (a doppia faccia)
- b) IBM—8 (a singola faccia)
- b1) IBM—8 (a doppia faccia)
- c) KayPro IV (a doppia faccia)
- d) KayPro II (a singola faccia)
- e) Osborne (a singola faccia)

Ciascuno di questi formati è compatibile con il sistema CP/M Commodore 128. Per ora, il sistema non può formattare questi tipi di dischetto ed usarli con buoni risultati nel sistema ospite (host), ma possono essere usati nel sistema C128. Questa parte del sistema è ancora in sviluppo.

La tabella di pagina seguente lista i parametri che questi formati di dischetto cercano quando leggono il terzo software party (comune) CP/M.

Produttori

	a	b	c	d	e	a1	b1
Tipo di Dischetto	**DSDD**	**SSDD**	**DSDD**	**SSDD**	**SSDD**	**DSDD**	**DSDD**
Posizione iniziale (t/s)	**2/1**	**1/1**	**0/10**	**1/0**	**3/1**	**2/1**	**1/1**
Misura del dischetto (in byte)	**256**	**512**	**512**	**512**	**1024**	**512**	**512**
Numero di settori/tracce	**32**	**8**	**10**	**10**	**5**	**20**	**8**
Numero di tracce	**40**	**40**	**80**	**40**	**40**	**40**	**80**
Misura allocaz. blocco	**2048**	**1024**	**2048**	**1024**	**1024**	**2048**	**2048**
# di entrate dirette	**128**	**64**	**128**	**64**	**64**	**128**	**64**
# di tracce riservate	**2**	**1**	**1**	**1**	**3**	**2**	**1**

SS = a singola faccia
DS = a doppia faccia
DD = a doppia densità

Tabella del formato dei Dischetti MFM

NOTA: Epson (a) etichetta i numeri di settore da 1 a 16. L'altro formato Epson QX10 etichetta i settori da 1 a 10. Sia l'alto che il basso del dischetto sono etichettati nello stesso modo. IBM (b) etichetta i numeri dei settori da 1 a 8. Sia l'alto che il basso del dischetto sono etichettati nello stesso modo. KayPro IV e KayPro II (c e d) etichettano i numeri dei settori da 0 a 9 in alto e da 10 a 19 in fondo (basso).

Questi valori sono presi dalla tabella MFM. Il vettore in $FD46 tiene il puntatore all'inizio della tabella etichettata MFM$table. Nel sistema corrente ci sono i formati letti e scritti compatibili per il sistema CP/M Plus del Commodore 128.

A pagina seguente diamo un listato della tabella del formato MFM.

```
db      S256*2+(16*2-8)+1          ;256 byte per settore,16 sett/traccia
db      MFM+S256+Type0+C0+S1       ;  DSDD
dw      0                          ;inizio su traccia 2 set 1(2 alc)
dpb     256,32,40,2048,128,2       ;sett# da 1 a 16
db      16                         ;(alto e basso hanno lo stesso numero)
db      'Epson QX10'               ;1 Epson QX10
db      80h+S512*2+(10*2-8)+1      ;512 byte per sett, 10tr/sett
db      S256*2                     ;traccia 0 è 256 byte/settore
db      MFM+S512+Type0+C0+S1       ;  DSDD
dw      0                          ;inizio traccia 2 sett 1 (2 alc)
dpb     512,20,40,2048,128,2       ;sett# da 1 a 10
db      10                         ;(alto e basso numerati nello stesso
                                     modo)
db      'Epson QX10'               ;2
db      S512*2+(8*2-8)+1           ;512 byte sett 8 sett/tr
db      MFM+S512+Type2+CO+S1       ;  SSDD
dw      0                          ;inizio traccia 1 sett 1 (2 alc)
dpb     512,8,40,1024,64,1         ;sett# da 1 a 8
db      8                          ;
db      'IBM-8 SS'                 ;3
db      S512*2+(8*2-8)+1           ;512 byte sett 8 sett/tr
db      MFM+S512+type2+C0+S1       ;  DSDD
dw      0                          ;inizio traccia 1 settore 1 (1 alc)
dpb     512,8,80,2048,64,1         ;sett# da 1 a 8
db      8                          ;(alto e basso sono numerati nello stesso
                                     modo)
db      'IBM-8 DS'                 ;4
db      S512*2+(10*2-8)+0          ;512 byte per settore, 10 sett/tr
db      MFM+S512+Type1+C1+S0       ;  DSDD
dw      0                          ;inizio traccia 0 settore 10 (2 alc)
dpb     512,10,80,2048,128,1       ;sett# da 0 a 9 in cima (tracce pari)
db      10                         ;sett# da 10 a 19 in basso (tracce dispa-
                                     ri)
db      'KayPro IV'                ;5
db      S512*2+(10*2-8)+0          ;512 byte per sett, 10 sett/tr
db      MFM+S512+Type0+C1+S0       ;  SSDD
dw      0                          ;inizio traccia 1 settore 0 (4 alc)
dpb     512,10,40,1024,64,1        ;sett? da 0 a 9
db      10                         ;
db      'KayPro II                 ;6
db      S1024*2+(5*2-8)+1          ;1024 byte per sett, 5 sett/tr
db      MFM+S1024+Tvpe0+C0+S1      ;  SSDD
dw      0                          ;inizio traccia 3 settore 1 (2 alc)
dpb     1024,5,40,1024,64,3        ;sett# da 1 a 5
db      5                          ;
db      'Osborne DD'               ;7
```

Tabella del Formato MFM

APPENDICE K - PARTE III

LA MAPPA DELLA MEMORIA DEL SISTEMA CP/M

Le pagine seguenti contengono la mappa della memoria del CP/M Z80 per il Commodore 128. La mappa della memoria comprende tutte le locazioni dei tasti CP/M, i vettori e le tabelle delle variabili. Utilizzatela come guida per il sistema CP/M dello Z80 all'interno del Commodore 128.

La mappa della memoria del CP/M è disponibile come file "CXEQU.LIB" del dischetto sul dischetto che si compra con il computer.

```
$*MACRO

false                   equ     0
true                    equ     non falso
banked                  equ     vero
EXTSYS                  equ     falso   ;usa il sistema esterno come dischetto e carattere I/O
pre$release             equ     falso
; inizio al 1 genn. 1978        78  79  80  81  82  83  84
dt$hx$yr                equ     365+365+366+365+365+365+366
;       1985                         1  2  3  4  5  6  7  8  9  10  11  12
date$hex                equ     dt$hx$yr+31+28+31+30+31+30+31+31+30+31+30+6
date    macro
        db              '6 Dic 85'
        endm
;
;
;
;
; boot della mappa della memoria (solo banco 0)
;
; bios02                equ      3000h  ;
block$buffer            equ      3400h  ;  usa 2K
boot$parm               equ      3C00h  ;  usa circa 256 bytes
;
;banco 0 mappa della memoria bassa
;
ROM                     equ      0000h
VIC$color               equ      1000h  ;  solo I/O pagina (selezionato IO$0)
SYS$key$area            equ      1000h  ;  3 blocchi di 256 byte (permessi 4)
screen$40               equ      1400h  ;  2x80x25=4000
BANK$parm$blk           equ      2400h  ;  permessi 0.5K di parametri
BIOS8502                equ      2600h  ;  1.5K
VIC$screen              equ      2C00h  ;  1K
ccp$buffer              equ      3000h  ;  0c80h (permessi 4)
bank0$free              equ      4000h  ;  inizio area libera nel banco 0
;
; locazioni di I/O in mappa
;
VIC                     equ     0D000h          ;
SID                     equ     0D400h          ;
MMU                     equ     0D500h          ;
DS8563                  equ     0D600h          ;  8563
VIC$C$H                 equ     0D800h          ;  (memoria a mappa solo in IO$0)
VIC$C$L                 equ     01000h          ;  (memoria e i/o a mappa in IO$0)
CIA1                    equ     0DC00h          ;  6526
CIA2                    equ     0DD00h          ;  6526
USART                   equ     0DE00h          ;  6551 (scheda esterna)
RAM$dsk$base            equ     0DF00h          ;  8726

;
; Allocazione della memoria comune
;
int$block               equ      0FC00h  ;  puntatori di interruzione del modo 2 (a FDFDh)
parm$block              equ      0FD00h  ;  parametri di sistema
@buffer                 equ      0FE00h  ;  buffer di disco (256 bytes)
;                                0FF00h  ;  a 0FFFFh usato con 8502
;
; le seguenti sono eguaglianze del systema C128
;
enable$z80              equ     0FFD0h          ;  codice 8502
return$z80              equ     0FFDCh
enable$6502             equ     0FFE0h          ;  codice Z80
return$6502             equ     0FFEEh
                                                ;  primo byte usato come puntatore di interruzione
vic$cmd                 equ     parm$block+1    ;;  byte di comando bios8502
vic$drv                 equ     vic$cmd+1       ;  drive bios8502 (mette bit 0, drv 0)
vic$trk                 equ     vic$drv+1       ;;  # della traccia bios8502

vic$sect                equ     vic$trk+1       ;;  # del settore bios8502
vic$count               equ     vic$sect+1      ;  conto del settore bios8502
vic$data                equ     vic$count+1     ;;  byte di dati verso/da bios8502
cur$drv                 equ     vic$data+1      ;  disco corrente installato nel Vir. drive
fast                    equ     cur$drv+1       ;  pone bit 0,drv 0 è veloce. ecc.

key$tbl                 equ     fast+1          ;;  puntatore alla tabella della tastiera
fun$tbl                 equ     key$tbl+2       ;;  puntatore alla tabella delle funzioni
color$tbl$ptr           equ     fun$tbl+2       ;;  puntatore alla tabella logica del colore
fun$offset              equ     color$tbl$ptr+2 ;;  # funzione da preformare
sound$l                 equ     fun$offset+1    ;;  non usato
sound$2                 equ     sound$1+2       ;;
sound$3                 equ     sound$2+2       ;;
```

```
@trk                equ   sound$3+2            ;;  numero di traccia corrente
@dma                equ   @trk+2               ;;  indirizzo del DMA corrente
;
;  le seguenti non usate dalle ROM
;
@sect               equ   @dma+2               ;   numero del settore corrente
@cnt                equ   @sect+2              ;   conto dei record per il transfer multisettore
@cbnk               equ   @cnt+1               ;   banco per le operazioni del processore
@dbnk               equ   @cbnk+1              ;   banco per le operazioni DMA
@adrv               equ   @dbnk+1              ;   drive del disco scelto correntemente
@rdrv               equ   @adrv+1              ;   controllore relativo al drive del disco
ccp$count           equ   @rdrv+1              ;   numero di records nel CCP
stat$enable         equ   ccp$count+1;         :   abilita lo stato di linea
                                               ;   7 kybrd, codici di tasto(1), funzioni(0)
                                               ;   tracciamento 6 40 colonne on(0),off(1)
                                               ;   stato del disco 0,abilita(1),disabilita(0)
emulation$adr       equ   stat$enable+1        ;   indirizzo dell'emulazione corrente
usart$adr           equ   emulation$adr+2      ;   puntatore al registro USART (6551)
;   eguaglianze CXIO
int$hl              equ   usart$adr+2          ;   interrompe la locazione arresto HL
int$stack           equ   int$hl+2+20          ;   usato correntemente solo 5*2
user$hl$temp        equ   int$stack            ;   locazione arresto HL delle funzioni utente
hl$temp             equ   user$hl$temp+2       ;   memorizzazione del temp misc (usato da VECTOR)
de$temp             equ   hl$temp+2            ;   memorizz del temp misc(usato da VECTOR)
a$temp              equ   de$temp+2            ;   memorizz del temp misc (usato da VECTOR)
source$bnk          equ   a$temp+1             ;   movimento tra banchi da banco# sorgente
dest$bnk            equ   source$bnk+1         ;   movimento tra banchi in banco# di destinazione
MFM$tbl$ptr         equ   dest$bnk+1           ;   puntatore alla tabella MFM
; fine della prima release
prt$conv$l          equ   MFM$tbl$ptr+2
ptr$conv$2          equ   ptr$conv$1+2
key$FX$function     equ   prt$conv$2+2
XxD$config          equ   key$FX$function+2
                                               ;   bit 7 0 = non parità 1 = parità
                                               ;   bit 6 0 = segno/spazio 1 = dispari/pari
                                               ;   bit 5 0 = spazio/pari 1 = segno/dispari

                                               ;   bit 1 0 = 1bit di stop 1 = bit di stop
                                               ;   bit 0 0 = 7bit di dati 1 = 8bit di dati

RS232$status        equ   XxD$config+1         ;   bit 7, 1 =invia dati 0 =nessun dato
                                               ;   bit 6, 1 =invio di dati
                                               ;   bit 5, 1 =attiva recv que
                                               ;   bit 4 1=errore di parità
                                               ;   bit 3, 1=errore di trasmissione
                                               ;   bit 2, 1=recv over run (perdita di dati) ( non usato)
                                               ;   bit 1, 1=ricezione dati
                                               ;   bit 0, 1=byte di dati pronto

xmit$data           equ   RS232$status+1       ;   byte di dati da inviare
recv$data           equ   xmit$data+1          ;   byte di dati ricevuti

;
;  Le seguenti eguaglianze sono usate dal gestore della tastiera per l'interruzione pilotata
;

int$rate            equ   recv$data+1

;
;       Il primo byte è un puntatore nella tabella, il secondo fino al dodicesimo rappresentano
;       lo stato corrente della tastiera (attivo basso), NOTA:
;       solo corrente se il key$buffer non è pieno
;
key$scan$tbl        equ   int$rate+1

;
;       la tastiera registra sul buffer
;
key$buf$size        equ   8*2                  ;   deve essere un numero pari di byte
key$get$ptr         equ   key$scan$tbl+12
key$put$ptr         equ   key$get$ptr+2
key$buffer          equ   key$put$ptr+2
;
;       buffer ricev del software UART
;
RxD$buf$size        equ   64
RxD$buf$count       equ   key$buffer+key$buf$size
RxD$buf$put         equ   RxD$buf$count+1
RxD$buf$get         equ   RxD$buf$put+1
RxD$buffer          equ   RxD$buf$get+1
tick$vol            equ   RxD$buffer+RxD$buf$size
;                   equ   tick$vol+1
```

```
INT$vector              equ    OFDFDh              ;; contiene un JMP int$handler
                                                   ; (in comune)
;→40 column misc parm
temp$1                  equ    BANK$parm$blk       ;;
@off40                  equ    temp$1+2            ;;
cur$offset              equ    @off40              ;;
old$offset              equ    @off40+2            ;;
prt$flg                 equ    old$offset+1        ;;
flash$pos               equ    prt$flg+1           ;;
;
;→ memorizzazione della posizione e del colore a 40 colonne
paint$size              equ    flash$pos+2         ;;
char$adr$40             equ    paint$size+1        ;;
char$col$40             equ    char$adr$40+2       ;;
char$row$40             equ    char$col$40+1       ;;
attr$40                 equ    char$row$40+1       ;;
bg$color$40             equ    attr$40+1           ;;
bd$color$40             equ    bg$color$40+1       ;;
rev$40                  equ    bd$color$40+1       ;;
;
;→ Memorizzazione della posizione e del colore a 80 colonne
char$adr                equ    rev$40+1            ;;
char$col                equ    char$adr+2          ;;
char$row                equ    char$col+1          ;;
current$atr             equ    char$row+1          ;;
bg$color$80             equ    current$atr+1       ;;
char$color$80           equ    bg$color$80+1       ;;
;       ROM usa locazioni oltre questo punto
;
;→ Parametri emulazione
parm$base               equ    char$color$80+1
parm$area$80            equ    parm$base+2
    ;                   ds     2                   ; exec$adr 80 colonne
    ;                   ds     1                   ; row # 80 colonne
parm$area$40            equ    parm$area$80+3
    ;                   ds     2                   ; exec$adr 40 colonne
    ;                   ds     1                   ; row # 40 colonne
buffer$80$col           equ    parm$area$40+3
;
;→ Parametri CXIO
;
;       int$count non è usato dalla release dello scorso 10 ottobre 85
int$count               equ    buffer$80$col+40*2          ; uno aggiunto ogni sessantesimo di secondo
key$buf                 equ    int$count+1
;
;→ Parametri CXKEYS
key$down$tbl            equ    key$buf+1                   ; non più usato (codice int)
;;;;; libero spazio sopra, il nuovo codice di interruzione pilotata non richiede questo spazio
;control$keys           equ    key$down$tbl+11*2; byte non più usato (codice int)

commodore$mode          equ    key$down$tbl+11*2
msgptr                  equ    commodore$mode+1
offset                  equ    msgptr+2
cur$pos                 equ    offset+1
string$index            equ    cur$pos+1
; Fine della prima releae (3 giugno 85)
sys$freq                equ    string$index+1              ; -1=50 Hz, 0= 60Hz
; Fine della seconda release (1 agosto 85)
;                       equ    sys$freq+1

;→ memorizzazione dei dati di temp del boot ROM
blk$ptr$cnt             equ    32
load$count              equ    boot$parm           ; numero di blocchi da 128 byte da caricare
ld$blk$ptr              equ    load$count+2        ; puntatore dma del settore corrente
blk$unld$ptr            equ    ld$blk$ptr+2        ; puntatore del blocco memoria di lettura(1k,2K)
block$size              equ    blk$unld$ptr+2      ; dimensione del blocco(1K=32 o 2K=64)
block$end               equ    block$size+1        ; permette di caricare 48K cpm.sys
block$ptrs              equ    block$end+2         ; fine del buffer di caricamento del blocco(+ 1K o + 2K)
info$buffer             equ    block$ptrs+blk$ptr$cnt
                                                   ; CPM3.sys carica indirizzi e conteggi
ext$num                 equ    info$buffer+12      ; CPM3.sys corrente ext #
retry                   equ    ext$num+1
boot$stack              equ    retry+1+64          ; permette 64 bytes di stack
;                       equ    boot$stack
;→ eguaglianze speciali usate dal CXKEY
special                 equ    00010111b
SF$exit                 equ    001h                ; TASTO RETURN
SF$insert               equ    028h                ; TASTO PIÙ
SF$delete               equ    02Bh                ; TASTO MENO
alpha$toggle            equ    03Dh                ; tasto commodore
alt$key                 equ    050h                ; tasto alterante
```

```
SF$left              equ   055h          ; tasto freccia sinistra
lf$arrow             equ   055h          ; tasto freccia sinistra
SF$right             equ   056h          ; tasto freccia destra
rt$arrow             equ   056h          ; tasto freccia destra

buff$large           equ   25
buff$small           equ   7
buff$pos             equ   7

;→ Controlli dell'interfaccia esterna RS232
rxd$6551             equ   USART+0       ; legge
txd$6551             equ   USART+0       ; scrive
status$6551          equ   USART+1       ; legge
reset$6551           equ   USART+1       ; scrive
command$6551         equ   USART+2       ; legge/scrive
control$6551         equ   USART+3       ; legge/scrive
txrdy                equ   10h
rxrdy                equ   08h
cmd$init             equ   0bh           ; no parità, attiva txd + rxd, interrompe off
cntr$init$19200      equ   1Fh           ; 1 stop, 8 bit, 19200 baud
cntr$init$9600       equ   1Eh           ; 1 stop, 8 bit, 9600 baud (internamente)
cntr$init$600        equ   017h          ; 600 baud

;→ locazioni di gestione della memoria
mmu$start            equ   MMU
conf$reg             equ   MMU           ; 3eh
conf$reg$1           equ   MMU+1         ; 3fh
conf$reg$2           equ   MMU+2         ; 7fh
conf$reg$3           equ   MMU+3         ; 3eh
conf$reg$4           equ   MMU+4         ; 7eh
mode$reg             equ   MMU+5         ; blh
ram$reg              equ   MMU+6         ; 0bh 16K top Common
page$0$1             equ   MMU+7         ; 00h
page$0$h             equ   MMU+8         ; 01h
page$1$1             equ   MMU+9         ; 01h
page$1$h             equ   MMU+10        ; 01h
mmu$version          equ   MMU+11        ; ??h

enable$C64           equ   11110001b     ; FS=0
Z80$off              equ   10110001b     ; valore da scrivere per abilitare 8502
Z80$on               equ   10110000b
fast$rd$en           equ   z80$on+0      ; lettura seriale veloce
fast$wr$en           equ   z80$on+8      ; scrittura seriale veloce
common$4K            equ   09h           ; cima 4K comune
common$8K            equ   0ah           ; cima 8K comune
common$16K           equ   0bh           ; cima 16K comune

;→ mappe di preconfigurazione
force$map            equ   0ff00h
bank$0               equ   0ff01h        ; 3fh
bank$1               equ   0ff02h        ; 7fh
io                   equ   0ff03h        ; 3eh
io$0                 equ   0ff03h        ; 3eh
io$1                 equ   0ff04h        ; 7eh;

;→ eguaglianze del video 80 colonne
DS$index$reg         equ   DS8563
DS$status$reg        equ   DS8563
DS$data$reg          equ   DS8563+1
;→ puntatori del registro
DS$cursor$high       equ   14
DS$cursor$low        equ   15
DS$rw$ptr$high       equ   18
DS$rw$ptr$low        equ   19
DS$rw$data           equ   31
DS$color             equ   26
;→ bit di stato
DS$ready             equ   80h
DS$lt$pen            equ   40h
;→ configurazione della memoria video (16K) 0–3fffh
DS$screen            equ   0000h
DS$attribute         equ   0800h
DS$char$def          equ   2000h
;
;→ eguaglianze VIC
; colori vic
nero                 equ   0
bianco               equ   1
rosso                equ   2
azzurro              equ   3
porpora              equ   4
verde                equ   5
blu                  equ   6
giallo               equ   7
arancio              equ   8
```

```
marrone              equ   9
lt$red               equ   10
dark$grey            equ   11
med$gray             equ   12
lt$green             equ   13
lt$blue              equ   14
lt$grey              equ   15

RM$status              equ    RAM$dsk$base        ; registro a sola lettura
;    bit 7           Interrompe il sospeso se 1
;        6           Trasferisce completo se 1
;        5           Verifica gli errori di blocco se 1
;    nota:           bits 5-7 sono cancellati quando letti
;        4           128K se 0, 512K se 1
;        3-0         Versione #
;
RM$command             equ    RAM$dsk$base+1; l/s
;    bit 7           esecuzione per la configurazione corrente se impostata
;        6           riservato
;        5
;                    abilità l'auto ricarica se impostata (rimemorizza tutti i
;                    registri ai valori precedenti all'invio del comando
;                    altrimenti punta al prossimo byte di lettura/scrittura.)
;        4           disabilita il decodificatore FF00 se posto (fa l'operazione dopo il comando writen)
;        3,2         riservato
;        1,0         00 = transfer C128 → Ram Disk
;                    01 = Transfer C128 ← Ram Disk
;                    10 = swap     C128 <-> Ram Disk
;                    11 = Verifica C128 = Ram Disk
;
RM$128$low             equ    RAM$dsk$base+2;  l/s
;       bits da 0 a 7 dell'indirizzo C128
;
RM$128$mid             equ    RAM$dsk$base+3;  l/s
;       bits da 8 a 15 dell'indirizzo C128
;
RM$ext$low             equ    RAM$dsk$base+4;  l/s
;       bits da 0 a 7 dell'indirizzo di Ram Disk
;
RM$ext$mid             equ    RAM$dsk$base+5;  l/s
;       bits da 8 a 15 dell'indirizzo di Ram Disk
;
RM$ext$hi              equ    RAM$dsk$base+6;  l/s
;       bit 16 dell'indirizzo di Ram Disk nella versione 128K
;       bits da 16 a 18 dell'indir. di Ram Disk nella versione 512K
;
RM$count$low           equ    RAM$dsk$base+7 ;  l/s
;       byte basso del transfer count (bits 0-7)
;
RM$count$hi            equ    RAM$dsk$base+8;  l/s
;       byte alto del transfer count (bits 8-15)
;
RM$intr$mask           equ    RAM$dsk$base+9;  l/s
;    bit 7           1=abilita il chip di interruzioni
;    bit 6           1=abilita la fine del blocco di interruzioni
;    bit 5           1=abilita la verifica errori delle interruzioni
;
RM$control             equ    RAM$dsk$base+10;  l/s
;  bit 7,6               00    Incrementa entrambi gli indirizzi (default)
;                        01    Fissa l'indirizzo espansione
;                        10    Fissa l'indirizzo C128
;                        11    Fissa entrambi gli indirizzi
;

;→ eguaglianze CIA
Data$a                 equ    00h
Data$b                 equ    01h
Data$dir$a             equ    02h
Data$dir$b             equ    03h
timer$a$low            equ    04h
timer$a$high           equ    05h
timer$b$low            equ    06h
timer$b$high           equ    07h
tod$sec$60             equ    08h
tod$sec                equ    09h
tod$min                equ    0ah
tod$hrs                equ    0bh
sync$data              equ    0ch
int$ctrl               equ    0dh
cia$ctrl$a             equ    0eh
```

```
cia$ctrl$b              equ     0fh
CIA$hours               equ     CIA1+tod$hrs

key$row                 equ     CIA1+Data$a     ; output
key$col                 equ     CIA1+Data$b     ; input
VIC$key$row             equ     0d02fh          ; output


data$hi                 equ     4               ; RS232 data line HI
data$low                equ     0               ; RS232 data line LOW
lf$shift$key            equ     80h
rt$shift$key            equ     10h
commodore$key           equ     20h
control$key             equ     04h


type$lower              equ     0
type$upper              equ     1
type$shift              equ     2
type$cntrl              equ     3
type$field              equ     00000011b


bnkl            equ     1
page0           equ     0
page1           equ     1

MMU$tbl$M               macro
        db              3fh,3fh,7fh,3eh,7eh             ; configurazione del registro
        db              z80$on,common$8K                ; mode & mem
        db              page0,bnk1,page1,bnk1           ; pagina del registro
        endm
;
;       funzioni ROM
;
TJMP    macro x
        rst 22 ! db x
        endm

TCALL   macro x
        mvi 1,x ! rst 4
        endm

RJMP    macro x
        rst 3 ! db x
        endm

RCALL   macro x
        mvi 1,x ! rst 5
        endm

FR$40                   equ     2       ; offset a funzioni ROM A 40 colonne


FR$wr$char              equ     00h     ; D=char auto avanzamento
FR$cursor$pos           equ     04h     ; B=riga C=colonna
FR$cursor$up            equ     08h
FR$cursor$down          equ     0Ch
FR$cursor$left          equ     10h
FR$cursor$rt            equ     14h
FR$do$cr                equ     18h
FR$CEL                  equ     1Ch
FR$CES                  equ     20h
FR$char$ins             equ     24h
FR$char$del             equ     28h
FR$line$ins             equ     2Ch
FR$line$del             equ     30h
FR$color                equ     34h     ; B=colore
FR$attr                 equ     38h     ; B=bit per mettere/cancellare, C=valore del bit
FR$rd$chr$atr           equ     3Ch     ; in D=riga, E=colonna
                                        ; out H=riga, L=colonna,B=char, C=attr(reale)
FR$wr$chr$atr           equ     40h     ; in D=riga, E=colonna, B=char, C=attr(reale)
                                        ; out H=riga, L=colonna
FR$rd$color             equ     44h
;FR$wr$color            equ     48h
;                       equ     4Ch
```

```
;
FR$trk$sect                     equ     50h
FR$check$CBM                    equ     52h
FR$bell                         equ     54h
;                               equ     56h
;                               equ     58h
;                               equ     5Ah
;                               equ     5Ch
;                               equ     5Eh

FR$trk$40                       equ     60h
FR$set$cur$40                   equ     62h
FR$line$paint                   equ     64h
FR$screen$paint                 equ     66h
FR$prt$msg$both                 equ     68h
FR$prt$de$both                  equ     6Ah
FR$update$it                    equ     6Ch
;                               equ     6Eh

FR$ASCII$to$pet                 equ     70h
FR$cur$adr$40                   equ     72h
FR$cur$adr$80                   equ     74h
FR$look$color                   equ     76h
;                               equ     78h
FR$blk$fill                     equ     7Ah     ; HL trasmesso allo stack
FR$blk$move                     equ     7Ch     ; '
FR$char$inst                    equ     7Eh     ; '

;
;       locazioni fissate in ROM
;
R$cmp$hl$de                     equ     100h-6
R$write$memory                  equ     180h+0
R$read$memory                   equ     180h+3
R$set$update$adr                equ     180h+6
R$wait                          equ     180h+9

R$status$color$tbl              equ     1000h-246-16
R$color$convert$tbl             equ     1000h-230-16
;
;       Definizione del byte di tipo di disco
;
;           bit     7               0=GCR,  1=MFM
;
;               se il bit 7 è 1 (MFM)
;                6          C0=0,  C1=1  (lato 2#, 0 fino (n/2)-1 o n/2 fino a n-1)
;                5,4        00=128, 01=256, 10=512, 11=1024 byte/settore
;                3,2,1      tipo di disco (MFM)
;                0           # del settore di partenza (0 o 1)
;
;               se il bit 7 è 0 (GCR)
;                6          non usato (posto a 0)
;                5,4        sempre 01 (settori a 256 byte)
;                3,2,1      tipo di disco (GCR)
;                           Tipo 0=nessuno, pone traccia e settore come impostati
;                           Tipo 1=tipo di disco C64
;                           Tipo 2=tipo di disco C128
;                0          non usato (posto a 0)

MFM         equu    1*128
C0          equ     0*64            ; secondo lato parte dall'inizio
C1          equ     1*64            ; secondo lato continua dal primo
C1$bit      equ     6
Type0       equ     0*2             ; (MFM) cima, fondo se è la prossima traccia
                                    ;           (TRK# 0 a (34 o 39))
Type1       equ     1*2             ; (MFM) cima (trk 0 pari), fondo (trk 1 dispari)
                                    ;           (TRK# 0 a (69 o 79))
Type2       equ     2*2             ; (MFM) cima TRK# 0 a 39, fondo TRK# 40 a 79
                                    ;           (TRK# su partenza inversa a 39 e va a 0)

Type7       equ     7*2             ; (MFM) passa i valori di byte forniti in 'trk
                                    ;           e 'sect
TypeX       equ     7*2
S0          equ     0*1             ; parte al settore 0
S1          equ     1*1             ; parte al settore 1
S128        equ     0*16
S256        equ     1*16
S512        equ     2*16
S1024       equ     3*16
```

```
;
dsk$none        equ   Type0+S256        ; accesso a qualsiasi settore del disco
dsk$c64         equu  Type1+S256
dsk$c128        equ   Type2+S256

dir$track       equ   18                ; disk dir track del C64;

;
;    comandi 6510
;
vic$reset       equ   -1       ; reboot C128
vic$init        equ   0        ; inizializza il bios8502
vic$rd          equ   1        ; legge un settore di dati (256 byte)
vic$wr          equ   2        ; scrive un settore di dati
vic$rdF         equ   3        ; predispone per rapida lettura (molti settori)
vic$wrF         equ   4        ; predispone per rapida scrittura
vic$test        equ   5        ; prova il disco corrente nel drive
vic$query       equ   6        ; fa partire i settori e # di sett/traccia
vic$prt         equ   7        ; stampa i caratteri dati
vic$frmt        equ   8        ; formatta un disco (1541)
vic$user$fun    equ   9
vic$RM$rd       equ   10       ; legge la RAM del disco
vic$RM$wr       equ   11       ; scrive la RAM del disco

;
;    caratteri di controllo
;
eom             equ   00h
bell            equ   07h
bs              equ   08h
lf              equ   0ah
cr              equ   0dh
xon             equ   11h
xoff            equ   13h
esc             equ   1bh
```

003D	ALPHATOGGLE			0050	ALTKEY	FD43	ATEMP	240D	ATTR40
FF01	BANK0	4000	BANK0FREE	FF02	BANK1	FFFF	BANKED		
2400	BANKPARMBLK			240F	BDCOLOR40	0007	BELL	240E	BGCOLOR40
2416	BGCOLOR30	3000	BIOS02	2600	BIOS8502	0000	BLACK	0020	BLKPTRCNT
3C04	BLKUNLDPTR	3400	BLOCKBUFFER			3C07	BLOCKEND	3C09	BLOCKPTRS
3C06	BLOCKSIZE	0006	BLUE	0001	BNK1	3C00	BOOTPARM	3C77	BOOTSTACK
0009	BROWN	0008	BS	2420	BUFFER80COL			0019	BUFFLARGE
0007	BUFFPOS	0007	BUFFSMALL	0000	C0	0040	C1	0006	C1BIT
3000	CCPBUFFER	FD21	CCPCOUNT	2411	CHARADR	2409	CHARADR40	2413	CHARCOL
240B	CHARCOL40	2417	CHARCOLOR80			2414	CHARROW	240C	CHARROW40
DC00	CIA1	DD00	CIA2	000E	CIACTRLA	000F	CIACTRLB	DC0B	CIAHOURS
000B	CMDINIT	001F	CNTRINIT19200			0017	CNTRINIT600		
001E	CNTRINIT9600			FD0D	COLORTBLTR				
DE02	COMMAND6552			0020	COMMODOREKEY				
2488	COMMODOREMODE			000B	COMMON16K	0009	COMMON4K	000A	COMMON8K
D500	CONFREG	D501	CONFREG1	0502	CONFREG2	0503	CONFREG3	D504	CONFREG4
DE03	CONTROL6551			0004	CONTROLKEY	000D	CR	FD07	CURDRV
2402	CUROFFSET	248C	CURPOS	2415	CURRENTATR	0003	CYAN	000B	DARKGREY
0000	DATAA	0001	DATAB	0002	DATADIRA	0003	DATADIRB	0004	DATAHI
0000	DATALOW	0B1C	DATEHEX	FD45	DESTBNK	FD41	DETEMP	0012	DIRTRACK
D600	DS8563	0800	DSATTRIBUTE			2000	DSCHARDEF	001A	DSCOLOR
000E	DSCURSORHIGH			000F	DSCURSORLOW			D601	DSDATAREG
D600	DSINDEXREG	0014	DSKC128	0012	DSKC64	0010	DSKNONE	0040	DSLTPEN
0080	DSREADY	001F	DSRWDATA	0012	DSRWPTRHIGH			0013	DSRWPTRLOW
0000	DSSCREEN	D600	DSSTATUSREG			09FD	DTHXYR		
FD23	EMULATIONADR			FFE0	ENABLE6502	00F1	ENABLEC64	FFD0	ENABLEZ80
0000	EOM	001B	ESC	3C35	EXTNUM	0000	EXTSYS	0000	FALSE
FD08	FAST	00B0	FASTRDEN	00B8	FASTWREN	2406	FLASHPOS	FF00	FORCEMAP
0002	FR40	0070	FRASCIITOPET			0038	FRATTR	0054	FRBELL
007A	FRBLKFILL	007C	FRBLKMOVE	001C	FRCEL	0020	FRCES	0028	FRCHARDEL
0024	FRCHARINS	007E	FRCHARINST	0052	FRCHECKCBM	0034	FRCOLOR	0072	FRCURADR40
0074	FRCURADR80	000C	FRCURSORDOWN			0010	FRCURSORLEFT		
0004	FRCURSORPOS			0014	FRCURSORRT	0008	FRCURSORUP	0018	FRDOCR
0030	FRLINEDEL	002C	FRLINEINS	0064	FRLINEPAINT				
0076	FRLOOKCOLOR			006A	FRPRTDEBOTH				
0068	FRPRTMSGBOTH			003C	FRRDCHRATR	0044	FRRDCOLOR		
0066	FRSCREENPAINT			0062	FRSETCUR40	0060	FRTRK40	0050	FRTRKSECT
006C	FRUPDATEIT	0000	FRWRCHAR	0040	FRWRCHRATR	FD0F	FUNOFFSET	FD0B	FUNTBL
0005	GREEN	FD3F	HLTEMP	3C29	INFOBUFFER	FC00	INTBLOCK	2470	INTCOUNT

Variabili

000D	INTCTRL	FD27	INTHL	FD52	INTRATE	FD3D	INTSTACK	FDFD	INTVECTOR
FF03	IO	FF03	IO0	FF04	IO1	2471	KEYBUF	FD63	KEYBUFFER
0010	KEYBUFSIZE	DC01	KEYCOL	2472	KEYDOWNTBL	FD4C	KEYFXFUNCTION		
FD5F	KEYGETPTR	FD61	KEYPUTPTR	DC00	KEYROW	FD53	KEYSCANTBL	FD09	KEYTBL
3C02	LDBLKPTR	000A	LF	0055	LFARROW	0080	LFSHIFTKEY	3C00	LOADCOUNT
000E	LTBLUE	000D	LTGREEN	000F	LTGREY	000A	LTRED	000C	MEDGRAY
0080	MFM	FD46	MFMTBLPTR	D500	MMU	D500	MMUSTART	D50B	MMUVERSION
D505	MODEREG	2489	MSGPTR	248B	OFFSET	2404	OLDFFSET	0008	ORANGE
0000	PAGE0	D508	PAGE0H	D507	PAGE0L	0001	PAGE1	D50A	PAGE1H
D509	PAGE1L	2408	PAINTSIZE	241D	PARMAREA40	241A	PARMAREA80	2418	PARMBASE
FD00	PARMBLOCK	FD48	PRTCONV1	FD4A	PRTCONV2	2405	PRTFLG	0004	PURPLE
DF00	RAMDSKBASE	D506	RAMREG	00FA	RCMPHLDE	0F0A	RCOLORCONVERTTBL		
FD51	RECVDATA	0002	RED	DE01	RESET6551	3C36	RETRY	FFEE	RETURN6502
FFDC	RETURNZ80	2410	REV40	DF02	RM128LOW	DF03	RM128MID	DF01	RMCOMMAND
DF0A	RMCONTROL	DF08	RMCOUNTHI	DF07	RMCOUNTLOW	DF06	RMEXTHI	DF04	RMEXTLOW
DF05	RMEXTMID	DF09	RMINTRMASK	DF00	RMSTATUS	0000	ROM		
0183	RREADMEMORY			FD4F	RS232STATUS				
0186	RSETUPDATEADR			0EFA	RSTATUSCO-LORTBL			0056	RTARROW
0010	RTSHIFTKEY	0189	RWAIT	0180	RWRITEMEMORY			DE00	RXD6551
FD73	RXDBUFCOUNT			FD76	RXDBUFFER	FD75	RXDBUFGET	FD74	RXDBUFPUT
003C	RXDBUFSIZE	0008	RXRDY	0000	S0	0001	S1	0030	S1024
0000	S128	0010	S256	0020	S512	1400	SCREEN40	002B	SFDELETE
0001	SFEXIT	0028	SFINSERT	0055	SFLEFT	0056	SFRIGHT	D400	SID
FD10	SOUND1	FD12	SOUND2	FD14	SOUND3	FD44	SOURCEBNK	0017	SPECIAL
FD22	STATENABLE	DE01	STATUS6551	248D	STRINGINDEX			000C	SYNCDATA
248E	SYSFREQ	1000	SYSKEYAREA	2400	TEMP1	0005	TIMERAHIGH	0004	TIMERALOW
0007	TIMERBHIGH	0006	TIMERBLOW	000B	TODHRS	000A	TODMIN	0009	TODSEC
0008	TODSEC60	FFFF	TRUE	DE00	TXD6551	0010	TXRDY	0000	TYPE0
0002	TYPE1	0004	TYPE2	000E	TYPE7	0003	TYPECNTRL	0003	TYPEFIELD
0000	TYPELOWER	0002	TYPESHIFT	0001	TYPEUPPER	000E	TYPEX	DE00	USART
FD25	USARTADR	FD3D	USERHLTEMP	D000	VIC	D800	VICCH	1000	VICCL
FD01	VICCMD	1000	VICCOLOR	FD05	VICCOUNT	FD06	VICDATA	FD02	VICDRV
0008	VICFRMT	0000	VICINIT	D02F	VICKEYROW	0007	VICPRT	0006	VICQUERY
0001	VICRD	0003	VICRDF	FFFF	VICRESET	000A	VICRMRD	000B	VICRMWR
2C00	VICSCREEN	FD04	VICSECT	0005	VICTEST	FD03	VICTRK	0009	VICUSERFUN
0002	VICWR	0004	VICWRF	0001	WHITE	FD50	XMITDATA	FD4E	XXDCONFIG
0007	YELLOW	00B1	Z80OFF	00B0	Z80ON	FD1F	@ADRV	FE00	@BUFFER
FD1D	@CBNK	FD1C	@CNT	FD1E	@DBNK	FD18	@DMA	2402	@OFF40
FD20	@RDRV	FD1A	@SECT	FD16	@TRK				

Variabili

APPENDICE L

SCHEMI CIRCUITALI DEL SISTEMA COMMODORE 128

Le otto pagine seguenti contengono tutti gli schemi circuitali del sistema Commodore 128. Ogni figura di due pagine rappresenta uno schema completo di ingegneria. Per una lettura più facile, il margine destro della pagina a sinistra ed il margine sinistro della pagina destra hanno parti dello schema che sono state duplicate. Questa sovrapposizione è stata fatta perchè possiate leggere il diagramma del circuito da entrambe le metà delle due pagine, poi passare alla pagina adiacente e riprendere dal punto lasciato dove finiva la pagina opposta. La freccia in cima ad ogni pagina fornisce una cornice di riferimento per segnare la parte del diagramma che è sovrapposto.

SCHEMA 310378 DEI MICROPROCESSORI, MMU, SID, PLA, ALIMENTAZIONE

SCHEMA 310378 DEL VIDEO, CLOCK, RESET

SCHEMA SC 310378 DI I/O – Porte di Controllo, Tastiera, Ingresso dell'Utente, Porta Seriale, Porta della Cassetta

8
7
6
5
D
C
B
A
TA12
128 / 64
OPEN · 128K ROM
CLOSE - 256K ROM
TA14
A0-A15
D0-D7
ROM1
ROM2
ROM3
ROM4
FROM1
74LS08
U8
R38
10K
C8
.1µF
J6
C32
.1µf
23128
23256
ROM1
ROM1/4
U32
(EPROM : PGM)
C33
C64
.01µf
23128
ROM2
U33
R36 10K
C34
J4
23128
23256
ROM3
ROM2/3
U34
C35
U35
VIC TOWER
74LS74A
FLAT PAC
-02 ONLY
-01 ONLY
DOT
RAS
DWE
GCAS0
CAS
GCAS1
R1
68Ω
U9
74 F 32 (FOR-01)
74 LS32 (FOR-02)
R29
68Ω
C9
.1µf
R30
68Ω
RAMCAS0
RAMCAS1
C38
.22µf
4164
U38
C39
.22µf
4164
U39
C40
.22µf
4164
U40
C121
.01µf
TA8-TA15
U62
74LS244
DMA
A8-A15
C46
.22µf
4164
U46
C47
.22µf
C65
.01µf
4164
U47
C48
.22µf
4164
U48
MA0 - MA7
D0- D7

SCHEMA 310378 RAM, ROM

GLOSSARIO

Questo glossario fornisce brevi definizioni dei termini di computeristica usati frequentemente.

Abilitare: Attivare un bit, byte o una operazione specifica del computer.

Acoustic Coupler or Acoustic Modem (*Accoppiatore Acustico* o *Modem Acustico*): Unità che converte i segnali digitali in suoni udibili per la trasmissione su linee telefoniche. La velocità è limitata a circa 1200 baud, o bit al secondo (bps). Confrontate con **Direct Connect Modem** (*Modem a Connessione Diretta*).

Address: Etichetta o numero che identifica il registro o la locazione di memoria dove viene memorizzata un'unità di informazione.

Alfanumerico: Lettere, numeri e simboli speciali che si trovano sulla tastiera, esclusi i caratteri grafici.

ALU: Unità Logica Aritmetica. La parte di un'Unità Centrale di elaborazione (CPU) dove vengono eseguite le operazioni matematiche.

Animazione: Uso delle istruzioni del computer per simulare il movimento di un oggetto sullo schermo con movimenti graduali e progressivi.

Array: Struttura di memorizzazione dei dati nella quale una serie di costanti o variabili connesse viene memorizzata in locazioni di memoria consecutive. Ogni costante o variabile contenuta in un array è considerata un elemento. Si accede ad un elemento usando un subscritto. Consultate **Subscritto.**

ASCII: Acronimo per American Standard Code for Information Interchange (*Codice Standard Americano per lo Scambio di Informazioni*), che è un codice di 7 bit usato per la rappresentazione dei caratteri alfanumerici. È un codice di comunicazione utile per cose come l'invio di informazioni da una tastiera al computer e da un computer all'altro. Consultate **Character String Code**.

Assembler: Programma che traduce le istruzioni del linguaggio assemblativo in istruzioni del linguaggio macchina.

Assembly Language (*Linguaggio Assemblativo*): Linguaggio orientato alla macchina nel quale la mnemonica è usata per rappresentare ciascuna istruzione in linguaggio macchina. Ogni CPU ha il suo linguaggio assemblativo specifico. Consultate **CPU** e **Linguaggio Macchina.**

Assignment Statement (*Istruzione di Assegnazione*): Istruzione BASIC che pone un elemento variabile, costante o array in un valore numerico o di stringa specifico.

Attack: Intervallo nel quale il volume di una nota musicale da zero raggiunge l'altezza massima.

Background Color (*Colore di Fondo*): Il colore della parte dello schermo in cui sono posti i caratteri.

Base di Dati, vedi **Database**

BASIC: Acronimo di Beginner's All-purpose Symbolic Instruction Code (*Codice per Principianti per le Istruzioni Simboliche per qualsiasi Scopo*).

Baud: Unità di velocità della trasmissione dei dati seriali. Il termine fu usato in origine per misurare la velocità della trasmissione via telegrafo. Trecento baud corrispondono approssimativam. alla velocità di trasmissione di 30 byte, o caratteri al secondo.

Binario: Un sistema numerico a base 2. Tutti i numeri sono rappresentati come una sequenza di 0 e 1.

Bit: Abbreviazione di Binary digIT (*cifra binaria*). Un bit è l'unità più piccola della memoria di un computer. Ogni cifra binaria può avere uno dei due valori, 0 o 1. Ci si riferisce ad un bit come posto o "on" se è uguale a 1. Un bit è

azzerato o "off" se è uguale a 0.

Bit Control (*Bit di Controllo*): Mezzo di trasmissione dei dati seriali nel quale ogni bit ha un significato ed un carattere singolo è circondato da bit di inizio o fine.

Bit di parità, vedi **Parity Bit**

Bit di partenza, vedi **Start Bit**

Bit di Stop, vedi **Stop Bit**

Bit Map Mode (*Modo Bit Map*): Modo grafico avanzato del Commodore 128 nel quale potete controllare ogni pixel dello schermo.

Border Color (*Colore del Margine*): Il colore dei margini intorno allo schermo.

Branch (*Salto*): Saltare ad una sezione di un programma ed eseguirla. GOTO e GOSUB sono esempi di istruzioni di salto in BASIC.

Bubble Memory (*Memoria a Bolle Magnetiche*): Un tipo relativamente nuovo di memoria del computer, usa piccole "tasche" o "bolle" magnetiche per memorizzare i dati.

Burst Mode: Un modo speciale ad alta velocità per la comunicazione tra un drive 1571 ed un computer C128, nel quale l'informazione viene trasmessa tante volte quanta è la velocità del Drive 1541.

Bus: Linee parallele o seriali usate per trasferire i segnali tra le unità. I computer vengono spesso descritti per la struttura del loro bus (cioè, computer a bus S-100, ecc.).

Bus Network (*Rete di Bus*): Sistema nel quale tutte le stazioni o le unità del computer comunicano usando un canale o bus a comune distribuzione.

Byte: Gruppo di 8 bit che costituiscono la più piccola unità di memorizzazione indirizzabile di un computer. Ogni locazione di memoria del Commodore 128 contiene 1 byte di informazione. Un byte è la unità di memorizzazione necessaria per rappresentare un carattere nella memoria. Consultate Bit.

Carattere: Qualsiasi simbolo della tastiera del computer che viene digitato sullo schermo. Il carattere include i numeri, le lettere, la punteggiatura ed i simboli grafici.

Caratteri di Grafica: Caratteri non alfanumerici sulla tastiera del computer.

Carrier Frequency (*Frequenza Portante*): Segnale costante trasmesso tra unità comunicanti che è modulato per codificare le informazioni binarie.

Cavo coassiale, vedi **Coaxial Cable**

Character Memory: (*Memoria del carattere*) Area della memoria del Commodore 128 che memorizza le forme dei caratteri codificati visualizzate sullo schermo.

Character Set (*Insieme di caratteri*): Gruppo di caratteri in relazione tra loro. Gli insiemi di caratteri del Commodore 128 consistono di lettere maiuscole, lettere minuscole e di caratteri grafici.

Character String Code (*Codice della Stringa di Caratteri*): Valore numerico assegnato per la rappresentazione di un carattere Commodore 128 nella memoria del computer.

Chip: Un circuito elettronico in miniatura che esegue le elaborazioni del computer come i grafici, i suoni e input/output.

Ciclo, vedi **Loop**

Ciclo di ritorno, vedi **Delay Loop**

Clock: Circuito di temporizzazione per un microprocessore.

Clocking: Tecnica usata per sincronizzare l'invio e la ricezione dei dati modulati per codificare le informazioni binarie.
Coaxial Cable (*Cavo Coassiale*): Mezzo di trasmissione di solito impiegato nelle reti locali.
Codice della Stringa di Caratteri, vedi **Character String Code**
Codice di schermo, vedi **Screen Code**
Codice Sorgente: Programma non eseguibile scritto in un linguaggio di livello superiore rispetto al codice macchina. Un compilatore o un assemblatore deve tradurre il codice sorgente in object code (linguaggio macchina) che il computer può capire.
Collision Detection (*Riconoscimento di Collisione*): Determina la collisione tra due o più sprite o tra sprite e dato.
Color Memory (*Memoria del Colore*): Area nella memoria del Commodore 128 che controlla il colore di ciascuna locazione nella memoria dello schermo.
Colore del margine, vedi **Border Color**
Colore di fondo, vedi **Background Color**
Comando: Istruzione in BASIC usata in modo diretto per eseguire una operzione. Consultate *Modo Diretto.*
Compilatore: Programma che traduce un linguaggio di livello superiore, come il BASIC, in linguaggio macchina.
Computer: Unità elettronica, digitale che mette in input, elabora e mette in output le informazioni.
Condizione: Espressione(i) tra le parole IF e THEN, in una frase IF... THEN, considerata vera o falsa. La frase condizionale IF...THEN permette al computer di prendere decisioni.
Contatore, vedi **Counter**
Coordinata: Punto singolo su una griglia che ha valori verticali (Y) ed orizzontali (X).
Counter (*Contatore*): Variabile usata per tenere il conto di quante volte è capitato un evento in un programma.
CPU: Acronimo di Central Processing Unit (*Unità Centrale di Elaborazione*), è la parte del computer che contiene i circuiti che controllano e compiono l'esecuzione delle istruzioni del computer.
Crunch: Minimizzare la quantità della memoria del computer usata per memorizzare un programma.
Cursore: Rettangolino che lampeggia che segna la locazione corrente sullo schermo.
Database (*Base di Dati*): Una grande quantità di dati in relazione tra loro memorizzati in modo ben organizzato. Un sistema di gestione del database è un programma che permette l'accesso alle informazioni
Data Link Layer (*Porzione di Controllo di Trasmissione*): Porzione logica del controllo della comunicazione dei dati che assicura che la comunicazione tra unità adiacenti è libera da errori.
Data Packet (*Pacchetto di Dati*): Mezzo di trasmissione dei dati seriali in un pacchetto efficiente che include una sequenza di controllo degli errori.
Data Rate o **Data Transfer Rate** (*Velocità di Trasferimento Dati*): La velocità con la quale i dati sono inviati ad un computer ricevente - data in baud, o bit al secondo (bps)
Datassette: Unità usata per memorizzare i programmi ed i file di dati in modo sequenziale su nastro.

Dato: Numeri, lettere o simboli che sono inviati in input dentro il computer e devono essere elaborati.

Debug: Correggere gli errori in un programma.

Decay: Intervallo nel quale il volume di una nota musicale decresce dal volume massimo ad un volume di mezzo chiamato valore sustain. Consultate *Sustain.*

Decrementare: Diminuire una variabile di indice o un contatore di un valore specifico.

Dedicated Line or **Leased Line** (*Linea Fissa*): Speciale disposizione di una linea telefonica fornita dalla compagnia telefonica e richiesta da certi computer o terminali, per cui il collegamento è sempre stabilito.

Delay Loop (*Ciclo di Ritardo*): Un ciclo vuoto FOR...NEXT che rallenta l'esecuzione di un programma.

Dial-Up Line (*Linea Commutata*): La normale linea telefonica commutata che può essere utilizzata come mezzo di trasmissione per la comunicazione dei dati.

Digitale: Di o connesso con la tecnologia della comunicazione dei computer e dei dati dove tutte le informazioni sono codificate come bit di 1 o 0 che rappresentano stati accesi o spenti.

Dimensione: La proprietà di un array (matrice) che specifica la misura e la direzione lungo un'asse nel quale gli elementi dell'array sono memorizzati. Per esempio, un array a due dimensioni ha un asse X per le righe e Y per le colonne. Consultate **Array.**

Direct Connect Modem (*Modem a Collegamento Rigido*): Unità che converte i segnali digitali di un computer in impulsi elettronici per la trasmissione sulle linee telefoniche. In contrasto con **Acoustic Coupler.**

Disabilitare: Disattivare un bit, byte o un'operazione specifica del computer.

Disk Drive: Unità di accesso casuale, con memoria di massa che salva e carica i file a/da un dischetto floppy (flessibile).

Disk Operating System (*Sistema Operativo del Dischetto*): Un programma usato per trasferire le informazioni per e da un dischetto. Chiamato spesso DOS.

Durata: La durata di tempo durante il quale una nota viene suonata.

Electronic Mail, or **E-Mail** (*Posta Elettronica*): Servizio di comunicazione per gli utenti del computer nel quale testi di messaggi vengono inviati ad un computer centrale, o ad una "cassetta della posta" elettronica e poi rintracciati dal destinatario.

EPROM: Un PROM che può essere cancellato dall'utente, di solito esponendolo ai raggi ultravioletti. Abbreviazione di Erasable Programmable Read Only Memory (*Memoria a Sola Lettura Programmabile e Cancellabile*). Consultate **PROM.**

Esadecimale: Riferito al sistema con base di 16 numeri. I programmi in linguaggio macchina sono spesso scritti in rappresentazione esadecimale.

Eseguire: Portare a termine delle istruzioni specificate in una istruzione di comando o di programma.

Espressione: Combinazione di costanti, variabili o elementi array elaborati da operatori logici, matematici o di confronto che ritornano un valore numerico.

File: Programma o insieme di dati trattati come un'unità e memorizzati su dischetto o nastro.

Firmware: Istruzioni del computer memorizzate nella ROM, come in una cartuccia di gioco.

Forma d'Onda: Rappresentazione grafica della forma di un'onda sonora. La forma d'onda determina alcune delle caratteristiche fisiche del suono.

Frequenza: Numero di onde sonore al secondo di un tono. La frequenza corrisponde all'altezza del tono udibile.

Frequenza portante vedi **Carrier Frequency**

Full-Duplex Mode (*Modo Pieno Duplex*): In questo modo, due computer possono trasmettere e ricevere i dati allo stesso tempo.

Funzione: Operazione predefinita che ritorna un singolo valore.

GCR: Abbreviazione di **Group Code Recording** (*Registrazione del Codice del Gruppo*), un metodo per memorizzare le informazioni su un dischetto. I drive 1541 e 1571 possono leggere, scrivere e formattare i dischetti GCR.

Generatore Envelope (*Generatore di Inviluppo*): Parte del Commodore 128 che produce forme d'onda specifiche (dente di sega, triangolo, ampiezza d'impulso e rumore) per le note musicali. Consultate **Waveform.**

Grafica: Immagini visive dello schermo che rappresentano dati del computer in memoria (cioè, caratteri, simboli e figure).

Grid (*Griglia*): Matrice a due dimensioni divisa in righe e colonne Le griglie sono usate per disegnare sprite e caratteri programmabili.

Griglia, vedi **Grid**

Half-Duplex Mode (Modo Semiduplex): In questo modo, i dati possono essere trasmessi solo in una direzione per volta; se un'unità sta inviando, l'altra deve semplicemente ricevere i dati finchè non è ora che trasmetta.

Hardware: Componenti fisiche del sistema di un computer, come la tastiera, i drive e la stampante.

Home: Angolo in alto all'estrema sinistra dello schermo.

IC: Abbreviazione di Integrated Circuit (*Circuito Integrato*). Un chip al silicone contenente un circuito elettrico fatto di componenti come transistor, diodi, resistori e condensatori. I circuiti integrati sono più piccoli e più efficienti dei circuiti individuali usati nei computer più vecchi.

Incrementare: Incrementare una variabile di indice o un contatore con un valore specificato.

Indice: Contatore variabile all'interno di un ciclo di programmazione.

Indirizzo, vedi **Address**

Input: Dato inserito nel computer per essere elaborato. Le sorgenti di input includono tastiera, drive, Datassette o modem.

Insieme di caratteri, vedi **Character set**

Interfaccia: Punto di incontro tra un computer ed un'entità esterna, sia un operatore, un'unità periferica o un mezzo di comunicazione. Una interfaccia può essere fisica se coinvolge un connettore o logica se coinvolge il software.

Intero: Numero intero (cioè,un numero che non contiene parti in frazioni), come 0, 1, 2, ecc.

I/O: Abbreviazione di Input/Output. Si riferisce al procedimento di inserire i dati nel computer, o di trasferire i dati dal computer ad un drive, stampante o mezzo di memorizzazione.

Istruzione di assegnazione, vedi **Assignement Statement**

Keybord (*Tastiera*): Componente di input di un sistema del computer.

Kilobyte (K): 1024 byte.
Linea Commutata, vedi **Dial-Upline**
Linea Fissa, vedi **Dedicated Line**
Linea di Programma: Frase o serie di frasi precedute da un numero di linea in un programma. La lunghezza massima di una linea di un programma del Commodore 128 è di 160 caratteri.
Linguaggio Macchina: Il linguaggio a più basso livello che il computer capisce. Il computer converte tutti i linguaggi ad alto livello, come il BASIC, in linguaggio macchina prima di eseguire qualsiasi istruzione. Il linguaggio macchina è scritto in forma binaria, che il computer può eseguire direttamente. Chiamato anche codice macchina o object code.
Local Network (*Rete Locale*): Uno dei numerosi schemi di comunicazione dei dati a breve distanza reso tipico dall'uso comune di un mezzo di trasmissione e da molte unità ad alta velocità dati. Chiamata anche Rete ad Area Locale, o LAN.
Locazione di Memoria: Indirizzo di memorizzazione specifico nel computer. Ci sono 131.072 locazioni di memoria (0-131.071) nel Commodore 128.
Loop (*Ciclo*): Segmento di programma eseguito ripetutamente per un numero specifico di volte.
Matrice: Rettangolo a due dimensioni con valori di riga e di colonna.
Memoria: Locazioni di memorizzazione dentro il computer. La ROM e la RAM sono due tipi diversi di memoria.
Memoria a bolle magnetiche, vedi **Bubble Memory**
Memoria del Carattere, vedi **Character Memory**
Memoria del Colore, vedi **Color Memory**
Memoria dello schermo, vedi **Screen Memory**
MFM: Abbreviazione di Modified Frequency Modulation (*Modulazione di Frequenza Modificata*), un metodo di memorizzazione delle informazioni su dischetto. I drive 1571 possono leggere e scrivere nei dischetti MFM.
Microprocessore: Un CPU contenuto in un singolo circuito integrato (IC). I microprocessori usati nei personal computer Commodore includono il 6510 l'8502 e lo Z80.
Mode (*Modo*): Uno stato di elaborazione.
Modem: Acronimo di **MOdulator/ DEModulator** (*MOdulatore/ DEModulatore*). Unità che trasforma i segnali digitali del computer in analoghi impulsi elettrici per la trasmissione su linee telefoniche e che fa l'inverso per la ricezione.
Modo Bit-Map Multicolore: Modo grafico che vi permette di visualizzare uno dei quattro colori per ogni pixel all'interno di una griglia di 8 per 8 caratteri. Consultate **Pixel.**
Modo Carattere Multicolore: Modo grafico che vi permette di visualizzare quattro diversi colori all'interno di una griglia di 8 per 8 caratteri.
Modo Carattere Standard: Il modo in cui il Commodore 128 opera quando lo accendete e quando scrivete i programmi.
Modo Diretto: Modo di esecuzione che esegue i comandi BASIC subito dopo che il tasto RETURN è stato premuto Chiamato anche Modo Immediato. Consultate **Comando.**
Monitor: Unità di visualizzazione che assomiglia ad un televisore ma con immagini a più alta risoluzione (più nette, nitide) sullo schermo.
Monitor Composto: Unità usata per fornire la visualizzazione del video a 40 colonne del C128.

Monitor RGBI: Unità di visualizzazione ad alta risoluzione necessaria per produrre il formato dello schermo ad 80 colonne del C128. RGBI sta per Red/Green/Blue/Intensity (Rosso/ Verde/Blu/Intensità).

Motherboard (*Scheda Madre*): In un sistema orientato al bus, la scheda che contiene le linee del bus ed i connettori per ricevere le altre schede del sistema.

Multiple-Access Network (*Rete ad Accesso Multiplo*): Sistema flessibile per il quale ogni stazione può accedere alla rete in qualunque momento; vengono presi dei provvedimenti per quelle volte in cui due computer decidono di trasmettere nello stesso momento.

Nota, vedi **Remark**

Null String (*Stringa Nulla*): Carattere vuoto (''). Un carattere che non è ancora assegnato ad un codice stringa del carattere. Produce una quantità non valida di errori se usata in una frase GET.

Operatore: Simbolo che dice al computer di eseguire un'operazione matematica logica o di confronto su variabili, costanti o elementi di array specificati nell'espressione. Gli operatori matematici sono +, –, *, /, e . Gli operatori di confronto sono <, =, >, < =, > = e <>. Gli operatori logici sono AND, OR NOT e XOR.

Ordine delle Operazioni: Sequenza con la quale vengono eseguiti i calcoli in un'espressione matematica. Chiamata anche Gerarchia delle Operazioni.

Ottava: Una serie completa di otto note di una scala musicale.

Parity Bit (*Bit di Parità*): 1 o 0 aggiunto ad un gruppo di bit che identifica la somma dei bit come dispari o pari, a scopo di controllo degli errori.

Parola, vedi **Word**

Periferica: Qualsiasi unità accessoria attaccata al computer tipo drive, stampante, modem o joystick.

Pitch: Altezza o bassezza di un tono che è determinata dalla frequenza dell'onda sonora. Consultate **Frequenza.**

Pixel: Termine del computer per picture element (*punto indirizzabile*). Ogni punto che forma un'immagine sullo schermo è chiamato pixel. Ogni carattere dello schermo viene spostato all'interno di una griglia di 8 per 8 pixel. L'intero schermo è composto da una griglia di 320 per 200 pixel. In modo bit-map, ogni pixel corrisponde ad un bit nella memoria del computer.

Polling (*Interrogazione Ciclica*): Un metodo di controllo delle comunicazioni usato da alcuni sistemi di computer/terminali per cui una stazione ''principale'' domanda a molte unità connesse ad un comune mezzo di trasmissione, a turno, se hanno informazioni da inviare.

Porta: Canale attraverso il quale i dati sono trasferiti a/da il CPU.

Porta in Parallelo: Porta usata per la trasmissione dei dati un byte per volta usando linee di 8 dati, una per ogni bit.

Porta Seriale: Porta usata per la trasmissione seriale dei dati; i bit vengono trasmessi uno per volta su un filo singolo.

Posta Elettronica, vedi **Electronic Mail**

Printer (*Stampante*): Unità periferica che mette in output i contenuti della memoria del computer su un foglio di carta. Ci si riferisce alla carta come hard copy (registrazione permanente).

Programma: Serie di istruzioni che portano il computer ad eseguire un compito specifico. I programmi possono essere memorizzati su disco o cassetta, risiedere nella memoria del computer, o essere listati su una stampante.

Programmabile: In grado di essere eseguito dalle istruzioni del computer.

PROM: Acronimo di Programmable Read Only Memory (*Memoria Solo Lettura Programmabile*). Chip semiconduttore della memoria i cui contenuti possono essere modificati.

Protocollo: Regole con cui i computer si scambiano le informazioni, compresa l'organizzazione delle unità dei dati da trasferire.

Puntatore: Registro usato per indicare l'indirizzo di una locazione in memoria.

Random Access Memory (RAM): Area programmabile della memoria del computer da cui si può leggere e in cui si può scrivere (cambiata). Tutte le locazioni della RAM sono accessibili nello stesso modo in qualsiasi minuto ed ordine. Il contenuto della RAM viene cancellato quando il computer viene spento.

Random Number (*Numero Casuale*): Numero decimale di 9 cifre da 0.000000001 a 0.999999999 generato dalla funzione RaNDom (RND).

Read Only Memory (ROM) (*Memoria a Solo Lettura*): Parte permanente della memoria del computer. Il contenuto delle locazioni ROM possono essere lette ma non cambiate. La ROM nel Commodore 128 contiene l'interprete del linguaggio BASIC, le forme dei caratteri delle immagini ed il sistema operativo.

Registro: Scomparti interni di memorizzazione con il microprocessore che comunicano tra i sistemi ROM, RAM e tra loro.

Release: Intervallo nel quale il volume di una nota musicale decresce dal valore sustain a 0.

Remark (*Nota*): Commenti usati per documentare un programma. Le note non sono eseguite dal computer, ma sono visualizzate nel listato del programma.

Rete ad accesso Multiplo, vedi **Multiple access network**

Rete ad anello, vedi **Ring Network**

Rete di Bus, vedi **Bus Network**

Rete locale vedi **Local Network**

Ribbon Cable (*Cavo Piatto*): Gruppo di fili attaccati in parallelo, di solito formati da 25 linee per la comunicazione RS-232.

Riconoscimento di Collisione, vedi **Collision Detection**

Ring Network (*Rete ad Anello*): Sistema nel quale tutte le stazioni sono collegate per formare un ciclo continuo o cerchio.

Risoluzione: La purezza del dettaglio di un'immagine visualizzata, determinata dalla densità dei pixel sullo schermo.

ROM, vedi **Read Only Memory**

RS-232: Standard consigliato di specifiche elettroniche ed elettromeccaniche per la comunicazione seriale. L'ingresso dell'utente in parallelo del C128 può essere trattato come porta seriale se vi si accede con il software, qualche volta con l'aggiunta di un'unità di interfaccia.

Salto, vedi **Branch**

Schermo: Unità di visualizzazione, che può essere o un televisore o un monitor video.

Screen Code (*Codice di Schermo*): Numero assegnato a rappresentare un carattere nella memoria dello schermo. Quando digitate un tasto sulla tastiera, il codice dello schermo di quel carattere è inserito nella memoria dello schermo automaticamente. Potete visualizzare un carattere anche memorizzando il suo codice di schermo direttamente nella memoria dello schermo con il comando POKE.

Screen Memory (*Memoria dello Schermo*): Area della memoria del Commodore 128 che contiene l'informazione visualizzata sullo schermo.
Sintassi: Regole grammaticali di un linguaggio di programmazione.
Sistema Operativo: Programma incorporato che controlla tutto quello che fa il computer.
Software: Programmi (insiemi di istruzioni) per il computer memorizzati su dischetto, nastro o cartuccia che possono essere caricati nella memoria ad accesso casuale (RAM). Il software, in breve, dice al computer cosa fare.
Sound Interface Device (SID): Chip sintetizzatore del suono sul MOS 6581 responsabile di tutte le funzioni audio del Commodore 128.
Sprite: Immagine grafica programmabile, mobile ad alta risoluzione. Chiamata anche Movable Object Block (MOB).
Stampante, vedi **Printer**
Start Bit (*Bit di Partenza*): Bit o gruppo di bit che identifica l'inizio di una parola di dati.
Statement: Istruzione in BASIC contenuta in una linea di programma.
Stop Bit (*Bit di Stop*): Bit o gruppo di bit che identifica la fine di una parola di dati e che definisce lo spazio tra le parole di dati.
Stringa: Carattere o serie di caratteri alfanumerici circondati da virgolette.
Subroutine: Segmento indipendente di programma separato dal programma principale che esegue un compito specifico. Le subroutine sono chiamate dal programma principale con la frase GOSUB e devono terminare con una frase RETURN.
Subscritto: Variabile o costante che si riferisce ad un elemento specifico di una matrice per mezzo della sua posizione nella matrice.
Sustain: Volume a metà intervallo di una nota musicale.
Tastiera, vedi **Keybord**
Tasti di Funzione: I quattro tasti all'estrema destra della tastiera del Commodore 128. Ogni tasto può essere programmato per eseguire una serie di istruzioni. Poichè i tasti possono essere traslati (SHIFTed), potete creare otto insiemi diversi di istruzioni.
Tono: Suono udibile di altezza e onda sonora specifiche.
Trasmissione Dati Asincrona: Schema di comunicazione in cui i caratteri dei dati vengono inviati ad intervalli di tempo, indipendenti dal clock del sistema. I limiti della trasmissione su linea telefonica sono di circa 2400 baud (bps). Consultate **Trasmissione Sincrona.**
Trasmissione Seriale: Invio di bit dei dati in ordine sequenziale.
Trasmissione Sincrona: Comunicazioni di dati usando un segnale di sincronizzazione o temporizzazione tra le unità invianti e quelle riceventi.
Trasparente: Descrive un'operazione del computer che non richiede l'intervento dell'utente.
Variabile: Unità di memorizzazione che rappresenta una stringa che cambia o un valore numerico. I nomi delle variabili possono avere qualsiasi lunghezza, ma sono memorizzati solo i primi due caratteri dal Commodore 128. Il primo carattere deve essere una lettera.
Velocità di Trasferimento Dati, vedi **Data Rate**
Verifica dell'errore o **Riconoscimento dell'errore:** Routine di software che identificano e spesso correggono i dati errati.
Video Interface Controller (VIC): Chip MOS (8564) responsabile delle funzioni grafiche a 40 colonne del Commodore 128.

Voce: Componente che produce il suono all'interno del chip SID. Ci sono tre voci all'interno del chip SID così il Commodore 128 può produrre tre suoni diversi nello stesso momento. Ogni voce consiste di un oscillatore del tono/generatore della forma d'onda, un generatore envelope ed un modulatore di ampiezza.

Word (*Parola*): Numero di bit trattati come una singola unità dal CPU. In un elaboratore ad 8 bit, la lunghezza della parola è di 8 bit; in un elaboratore a 16 bit, la lunghezza della parola è di 16 bit.

commodore 128

LA GRANDE GUIDA

AA.VV.

L'enciclopedia, nata con la collaborazione di tutti i tecnici responsabili della Commodore, si presenta come la guida di riferimento ufficiale ad uno dei più versatili e interessanti computer attualmente disponibili. Con 128K di memoria, espandibile fino a 640K, schermo a 80 colonne, compatibilità hardware e software col Commodore 64, funzionamento in modo CP/M 3.0, linguaggio BASIC espanso e molte altre nuove ed estese capacità e caratteristiche, il Commodore 128 non ha bisogno di presentazioni ma necessita solo di questa guida per essere usato in maniera ideale, sfruttando tutte le sue potenzialità e venendo a conoscenza di tutti i suoi più nascosti meccanismi.

- Fondamentali del BASIC
- Enciclopedia del BASIC 7.0
- Un passo oltre il normale BASIC
- Tecniche di programmazione avanzata per i modem Commodore
- Programmazione grafica del Commodore 128
- Linguaggio Macchina
- Come immettere programmi in Linguaggio Macchina
- Uso combinato di Linguaggio Macchina e BASIC
- La potenza nascosta della grafica
- Gli sprite
- Programmazione del chip a 80 colonne
- Suono e musica
- Guida all'input/output
- Il sistema operativo CP/M 3.0
- Mappe della memoria
- Specifiche hardware
- Appendici e glossario

GRUPPO EDITORIALE JACKSON

ISBN 88-7056-948-9

9 788870 569483

L. 50.000 Cod. CC750

705

commodore 128

LA GRANDE GUIDA